

Masters Program in **Geospatial Technologies**



**HETEROGENEOUS SENSOR DATABASE FRAMEWORK FOR THE SENSOR
OBSERVATION SERVICE: INTEGRATING REMOTE AND IN-SITU SENSOR
OBSERVATIONS AT THE DATABASE BACKEND**

MADUAKO IKECHUKWU

Dissertation submitted in partial fulfilment of the requirements
for the Degree of *Master of Science in Geospatial Technologies*

HETEROGENEOUS SENSOR DATABASE FRAMEWORK FOR THE SENSOR
OBSERVATION SERVICE: INTEGRATING REMOTE AND IN-SITU SENSOR
OBSERVATIONS AT THE DATABASE BACKEND

Dissertation supervised by

Prof. Dr. Edzer Pebesma

Co-supervised by

Dr. Ismael Sanz

&

Dr. Pedro Cabral

FEBUARY 2012

Declaration of Originality

This is to certify that this thesis is completely my own work with no plagiarism, unless clearly acknowledged (including citation of published and unpublished sources). The thesis has not been submitted before to other universities or to any other institution.

ACKNOWLEDGEMENT

With utmost sense of reverence, I want to say thanks to the European Commission and the distinguished persons organizing this prestigious Erasmus Mundus Programme from the Institute of Geoinformatics, university of Muenster, Universitat Jaume I, Spain and University of Lisbon, Portugal for this great opportunity given to me to participate in this wonderful master's programme.

I express sincere appreciation to Christoph Stasch, my advisor for his unquantifiable help and support to make this a success. Lots of thanks are due to Prof. Edzer Pebesma for his great inspiration and idea that informed this thesis.

Thanks are also due to my co-supervisors, Dr. Ismeal Sanz and Dr. Pedro Cabral from Universitat Jaume I Spain and University of Lisbon Portugal respectively for their support and contributions. I could not meet them more often within this thesis period because of distance but they gave me a solid 'go-ahead' support and inputs for this accomplishment.

I want express my gratitude to Jorge Arevalo, the PostGIS Raster developer for his great help during the system configuration for the prototypical implementation of the research.

And lastly I want to express my gratitude to my colleagues and students whose interaction contributed to the success of this work.

ABSTRACT

Environmental monitoring and management systems in most cases deal with models and spatial analytics that involve the integration of in-situ and remote sensor observations. In-situ sensor observations and those gathered by remote sensors are usually provided by different databases and services in real-time dynamic service systems like the Geo-Web Services. Thus, data have to be pulled from different databases and transferred over the web before they are fused and processed on the service middleware. This process is very massive and unnecessary communication and work load on the service, especially when retrieving massive raster coverage data. Thus in this research, we propose a database model for heterogeneous sensor-types that enables geo-scientific processing and spatial analytics involving remote and in-situ sensor observations at the database level of the Sensor Observation Service, SOS. This approach would be used to reduce communication and massive workload on the Geospatial Web Service, as well make query request from the user end a lot more flexible. Hence the challenging task is to develop a heterogeneous sensor database model that enables geo-processing and spatial analytics at the database level and how this could be integrated with the geo-web services to reduce communication and workload on the service and as well make query request from the client end more flexible through the use of SQL statements.

List of Figures

Figure 1 : The Conceptual Diagram.....	2
Figure 2: Workflow for comparing CIMIS and GOES temperature raster images	10
Figure 3: Overview of the communications flow and steps for the automated processing and calculation of vegetation productivity (Taken from (1)).....	11
Figure 4: Image showing WARMER information flows (taken from (13))	12
Figure 5: Sensor Web Concept	(Source of Image courtesy of the OGC). 15
Figure 6: Conceptual Coverage Diagram	21
Figure 7: UML Conceptual Model of the concept and Management of Coverages	22
Figure 8 : Regularly tiled rectangular raster coverage.....	26
Figure 9 : UML Conceptual Schema Model of the Proposed Heterogeneous Sensor Database..	31
Figure 10: ER-diagram and logical design of the database model.....	35
Figure 11: Conceptual Model of the proposed Web Query Service WQS.....	37
Figure 12: OpenJump Remote Database Query Connection Interface	37
Figure 13: Proposed Conceptual Architecture of Integrating the Heterogeneous Database and the Web Services	38
Figure 14: Multiple Database Approach	Figure 15: Common Database
Approach.....	41
Figure 16: Multiple Database Approach for scenario 3	47
Figure 17: Remote Sensor Land Surface Temperature LST Data	56
Figure 18: Remote Sensor Sea Surface Temperature SST Data.....	56
Figure 19: Remote Sensor Normalised Difference Vegetation Index NDVI Data	57
Figure 20: A screen shot excerpt of the heterogeneous sensor database model with the tables	59
Figure 21: Screen short excerpt of Scenario 1 implementation in OpenJump	61
Figure 22: Screen short excerpt of a sample Scenario 2.1 implementation result in OpenJump	62
Figure 23: Screen short excerpt of a sample Scenario 2.2 implementation result in OpenJump	64
Figure 24: Screen short excerpt of a sample Scenario 3 implementation result in OpenJump ...	65
Figure 25: Screen short excerpt of a sample Scenario 4 implementation result in OpenJump client end	67
Figure 26: Screen short excerpt of a sample Scenario 5 implementation result in OpenJump client end	68
Figure 27: Current workflow in a dynamic geo-processing service.....	71
Figure 28: Proposed workflow in a dynamic geo-processing service by means of the heterogeneous sensor database	72
Figure 29: Heterogeneous sensor database and Geo-web services integration model	76

List of Tables

Table 1: NOAA Air temperature table sample (Taken from: (23)).....	23
Table 2 : Evaluation Matrix between PostGIS Raster and Oracle GeoRaster Functionalities (taken from: (25))	28
Table 3: In-Situ Shipboard Sensor Air and Water Temperature Data	55
Table 4: In-Situ Sensor Land Surface Temperature LST Data, Daily Average	55

Listings

Listing 1: SQL sample query for scenario 1	41
Listing 2: SQL sample query for Scenario 2.1.....	42
Listing 3: SQL sample query for sample SQL queries Scenario 2.2	44
Listing 4: SQL sample query for Scenario 3.....	45
Listing 5: SQL sample query for Scenario 4.....	48
Listing 6: SQL sample query for abstracting maximum and minimum pixel values	49
Listing 7: SQL sample query for describing raster coverage in the database	50
Listing 8 : Scenario 1 implementation SQL code	60
Listing 9: Scenario 2.1 implementation SQL code	62
Listing 10: Scenario 2.2 implementation SQL code	63
Listing 11: Scenario 3 implementation SQL code	64
Listing 12: SQL Query to obtain the NDVI _{max}	66
Listing 14: Scenario 4 implementation SQL code	66
Listing 13: SQL Query to obtain the NDVI _{min}	66
Listing 15: Scenario 5 implementation SQL code	67

List OF Acronyms

AET - Actual Evapotranspiration
AFIS - Advanced Fire Information System
API - Application Programming Interface
ASCII – American Standard Code for Information Interchange
AWDN- Automated Weather Data Network
CIMIS - California Irrigation Management Information System
CSV - Comma Separated Values
ER – Entity Relationship
ESPG - European Petroleum Survey Group
FVC - Fraction of Vegetation Cover
GDAL- Geospatial Data Abstraction Library
Geotiff – Georeferenced Tagged Image File Format
GIS – Geographic Information System
GiST - Generalized Search Tree
GLCF - Global Land Cover Facilities
GML- Geographic Markup Language
GOES - Geostationary Operational Environmental Satellite
Hdf - Hierarchical Data Format
HPRCC - High Plains Regional Climate Center
HTTP - Hypertext Transfer Protocol
ISIES - Intelligent Sensorweb for Integrated Earth Sensing
ISO - International Organisation for Standards
Jpeg - Joint Photographic Expert Group (Image Format)
LST- Land Surface Temperature
MDD - Multi-Dimensional Discrete Data
MIS - Marine Information System
MODIS – Moderate Resolution Imaging Spectroradiometer
NASA - National Aeronautics and Space Administration
NDVI - Normalised Difference Vegetation Index
NEO - NASA Earth Observations
NOAA- National Oceanic and Atmospheric Administration
O&M - Observations & Measurements
OGC - Open Geospatial Consortium
Png - Portable Network Graphics
RasdaMan - Raster Database Manager
RasQL - Raster Data Query Language
RET - Referenced EvapoTranspiration
SensorML - Sensor Model Language
SOS - Sensor Observation Service
SPS - Sensor Planning Service
SQL – Structured Query Language
SRID - Spatial Reference Identifier
SRTM - Shuttle Radar Topographic Mission

SST - Sea Surface Temperature
SWE - Sensor Web Enablement
Tiff – Tagged Image File Format
TML - Transducer Markup Language
UML – Unified Modeling Language
USGS – United States Geological Survey
WCPS – Web Coverage Processing Service
WCS – Web Coverage Service
WFS – Web Feature Service
WKB - World Known Binary
WMS - Web Map Service
WNS - Web Notification Services
WQS – Web Query Service
XML- Extensible Markup Language

Table of Contents

ACKNOWLEDGEMENT.....	iv
ABSTRACT.....	v
List of Figures.....	vi
List of Tables.....	vii
Listings.....	viii
List OF Acronyms.....	ix
CHAPTER 1.....	1
1.0 INTRODUCTION	1
1.1 Research Problem:.....	3
1.2 Research Hypothesis:	3
1.3 Research Questions:	3
1.4 Methodology	3
1.5 Thesis Structure Overview.....	4
CHAPTER 2.....	5
2.0 BACKGROUND:	5
2.1 Studies dealing on In-Situ Sensor Databases:	5
2.2 Studies dealing on Remote Sensor (Raster) Databases:	6
2.3 Integrating Remote and In-situ Sensor Observations	9
2.4 Sensor Web Enablement, SWE	14
2.5 Sensor Observation Service SOS.....	15
2.6 Web Coverage Service WCS	16
2.6 Web Coverage Processing Service WCPS	17
CHAPTER 3.....	18
3.0 REQUIREMENT ANALYSIS	18
3.1 Datatype Analysis; Differences and Commonalities.....	18
3.2 Database Requirement Analysis for In-situ Sensor Observations.....	22
3.3 Database Requirement Analysis for Remote Sensor Observations (Raster data)	26
3.4 Spatial Database Management System Capability Analysis.....	27

CHAPTER 4.....	29
4.0 CONCEPTUAL DESIGN AND MODELLING OF THE DATABASE SCHEMA	29
4.2 The Heterogeneous Database Schema Entity Description.....	29
4.1 UML Conceptual Schema Model of the Proposed Heterogeneous Sensor Database	31
4.3 ER-diagram and logical design of the database model	34
4.4 Integrating the Heterogeneous Sensor Database with the OGC Web Services.....	36
CHAPTER 5.....	40
5.0 SCENARIO DESCRIPTION	40
5.1 Scenario 1: In-situ and satellite surface temperature analysis	40
5.2 Scenario 3: Geo-scientific Analysis of In-situ Temperature /Air Pressure Data and Satellite Elevation/ Height Observation	44
5.3 Scenario 4: Estimation of Actual Crop Evapotranspiration ET	46
5.4 Scenario 5: DescribeCoverage (Raster Coverage Metadata) Query Operation.	49
CHAPTER 6.....	51
6.0 POSTGIS-BASED PROTOTYPICAL IMPLEMENTATION, PERFORMANCE AND HYPOTHESIS EVALUATION.....	51
6.1 System Configuration	51
6.2 Test Data Collection and Description	54
6.3 Physical Database Model Design	57
6.3 Implementation of some of the Application Scenarios and Use Cases.....	59
6.4 Performance and Hypothesis Evaluation	68
6.3.1 Performance Analysis	68
6.3.2 Research Hypothesis Evaluation	70
Chapter 7.....	74
7.0 CONCLUSION AND FUTURE WORK	74
References	77
Appendixes.....	79

CHAPTER 1

1.0 INTRODUCTION

Geo-sensors gathering data to the geospatial sensor web can be classified into remote sensors and in-situ sensors. Remote sensors include satellite sensors, UAV, LIDAR, Aerial Digital Sensors (ADS) and so on measuring environmental phenomena remotely. These sensors acquire data in raster format at larger scales and extent. In-situ sensors are spatially distributed sensors over a region used to monitor and observe environmental conditions such as temperature, sound intensity, pressure, pollution, vibration, motion etc. These sensors are measuring phenomena in their direct environment and could be said to acquire data in vector or feature data format.

Most environmental monitoring and management systems combine these diverse datasets from heterogeneous sensors for environmental modeling and analysis. For example monitoring and estimating of actual Evapotranspiration at a particular location involves the fusion of different sensor data. This involves the aggregation of fraction of vegetation cover which is derived from Normalised Difference Vegetation Index, NDVI generated from coverage data and the gridded reference Evapotranspiration map derived from automatic weather stations. Remote and in-situ sensor data aggregation is also needed in dynamic web mapping services for vegetation productivity (1) or in marine information system (2) .

Meanwhile the fusion and aggregation of these sensor data on the web service currently involves massive data retrieval from different sensor databases, most especially from the raster databases, geo-processing and spatial analytics on service middleware. For web services this is massive work and communication load over the network and on the service. A sensor database management system combining remote and in-situ observations would be of high benefit for environmental monitoring and management systems. Having these disparate data on one database schema would leverage geospatial web services from excessive work load or data transfer through the network. Most of the data fusion, aggregations and processing done by web services can now be carried out at the database backend and the results retrieved through the appropriate web services to user end. Geo-scientific queries involving information from raster and vector data is possible with such a database. For example a scenario where a geo-

scientist want to compare for example the surface temperature value of a particular location from satellite observation and from in-situ temperature observations can easily be realized with this approach at the database backend without having to retrieve the massive raster data. Developing the database model that integrates remote sensor and in-situ sensor observations to leverage some of the above mentioned geo-scientific scenarios on the geospatial web services is the challenge and the essence of this research.

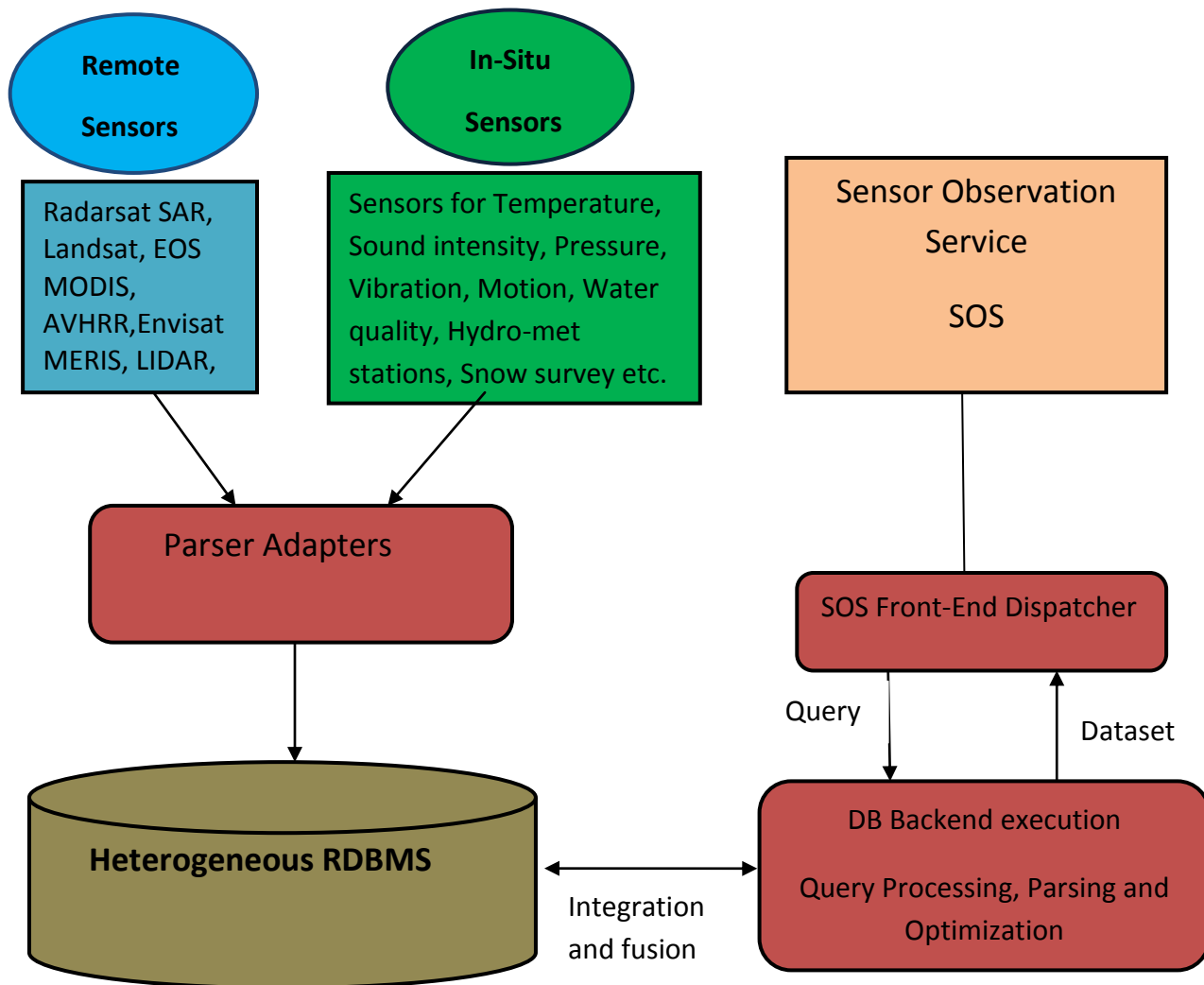


Figure 1 : The Conceptual Diagram

Figure 1 describes the research concept diagrammatically, showing the in-situ and remote sensor data passed to the heterogeneous sensor database where data integration , fusion and processing are carried out within the Sensor Observation Service, SOS.

1.1 Research Problem:

The research problem for this thesis is the development of a heterogeneous sensor database model that enables Geo-scientific queries involving remote and in-situ sensor data aggregation at the database level and how this can be leveraged to reduced excessive work and communication load for relevant Geospatial Web Services.

1.2 Research Hypothesis:

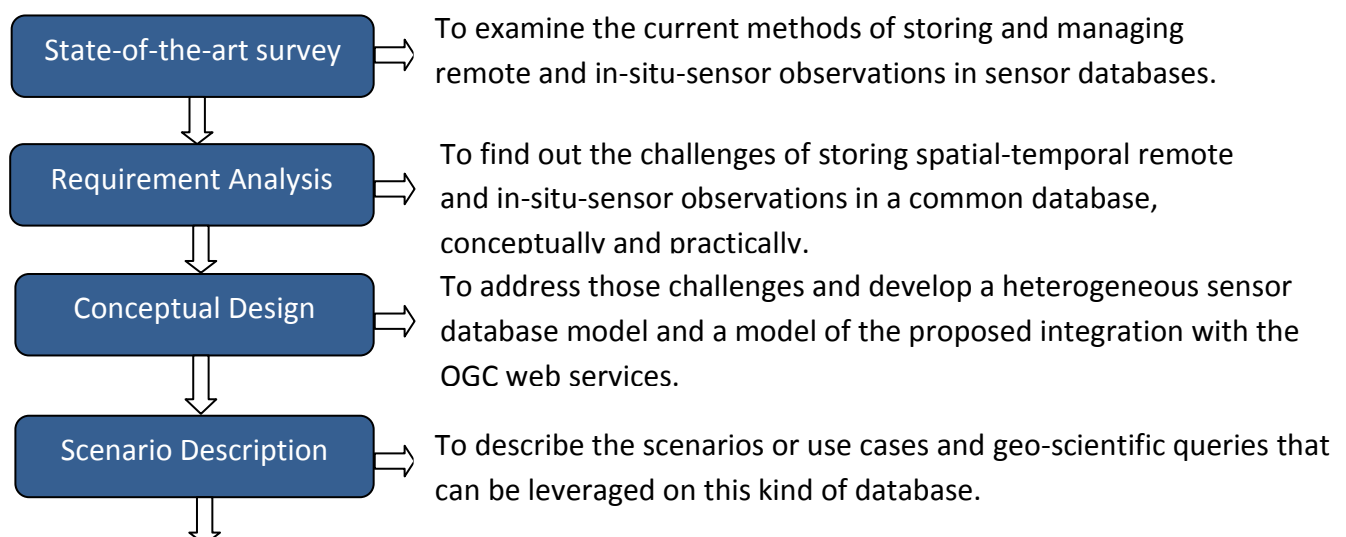
If remote and in-situ sensor observations can be effectively integrated at the database backend, then communication and work load on the service middleware of the Geo-Web Services can be drastically reduced and query requests from the users would be more flexible.

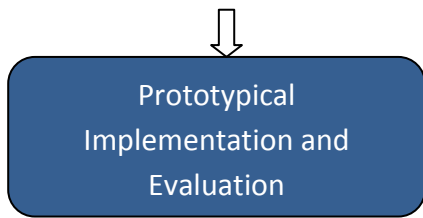
1.3 Research Questions:

- What are the methods that can be adopted for modeling heterogeneous sensor observations for databases?
- What are the capabilities of existing DBMS in handling spatio-temporal and heterogeneous data models?
- What are the specific constraints in implementing database schema that can be used for heterogeneous spatio-temporal observations?
- How can geo-spatial web services leverage this solution in reducing excessive work and communication load on the service?

1.4 Methodology

To answer the above mentioned research questions towards solving the research problem and evaluate the research hypothesis, we approached this research in the following systematic sequence.





And finally is the development of a prototypical implementation of this database model to evaluate and demonstrate the approach.

1.5 Thesis Structure Overview

The rest of this thesis is organised as follows: Chapter 2 provides the background work of existing studies and projects involving in-situ sensor observation database, raster database, integration of remote and in-situ sensor observations and also the OGC Sensor Web Enablement and some its specifications. Chapter 3 provides the database system requirement analysis for the seamless integration of in-situ and remote sensor observations at the database backend. Chapter 4 provides the conceptual and logical models of the proposed heterogeneous sensor database model and explains how the entities of the database schema relate. It also provides and discusses the conceptual model of how the heterogeneous database model can be integrated with other geo-web services for effectively services to the clients. Chapter 5 describes some of the scenarios and use cases where this type sensor database can be efficiently leveraged for geo-scientific analysis and processes. Chapter 6 provides the PostGIS-based prototypical implementation of the proposed heterogeneous sensor database model and provides the performance analysis of this prototypical implementation. In this section also is the discussion and evaluation of the research hypothesis based on the performance and results. Chapter 7 summarises this research and discusses the future work which will lead to more appreciation of this research work.

CHAPTER 2

2.0 BACKGROUND:

Existing works and studies in in-situ sensor observations database and in remote sensor (raster) database provide insight into the possibility of developing a database models to integrate the two disparate data types. These studies offer a theoretical background for this.

The literature review or background to this thesis can be divided into the following categories; studies related to in-situ sensor observations databases, studies related to remote sensor observations (raster) databases, studies related to the integration of remote and In-situ sensor observations and finally OGC Sensor Web Enablement (SWE), Sensor Observation Service (SOS), Web Coverage Service WCS and Web Coverage Processing Service WCPS.

2.1 Studies dealing on In-Situ Sensor Databases:

The current Sensor Observation Service SOS of 52 North organization is an in-situ sensor observation service implemented as a servlet and can be deployed in any servlet container, such as Apache Tomcat. In-situ sensor observations are stored in the PostgreSQL database. Users can create customized SQL query tables with new data and metadata. On the client side, the OX-Framework provides the access to SWE SOS and subsequent visualization of the queried data from the database. The 52 North in-situ sensor observation data service can be implemented in a broad environmental monitoring and management areas. For example, it has been used in the implementation in the AFIS (Advanced Fire Information System) (3). However the shortcoming of this approach is that is a single type sensor database service, optimized for In-situ sensors. When dealing with a project that requires integration with coverage data, the coverage data will have to be pulled from a different raster database, e.g. implemented by a Web Coverage Service (4). This is not optimal for an application using these data because usually the coverage data retrieval and processing is very massive over the network.

The SOS of the organization known as Deegree is another in-situ observation data service that is designed and implemented to connect to PostGIS, Oracle Spatial, or any database management system supporting JDBC. This supports the GetCapabilities, DescribeSensor, DescribePlatform and GetObservation operations.

Deegree SOS is very configurable enabling existing relational databases to be connected. On the client side, iGeoPortal is the web-based portal framework and offers visualization of geodata through a standard web browser (3). This is another in-situ sensor data type SOS without the possibility of coverage and feature data aggregation at the database backend.

2.2 Studies dealing on Remote Sensor (Raster) Databases:

Why Raster Databases?

Peter Baumann et al. (5) stressed the importance of efficient raster data storage and retrieval, as raster data acquisition is increasingly becoming easier and less expensive. Currency and large area coverage of raster imageries are substantially higher when compared to vector data. Hence raster data competes and complements favorably with vector data in most geo-application. Consequently the need for efficient, flexible large scale raster data storage coupled with remote access and retrieval is very paramount.

The authors (5) also pointed out the advantages of a database system for raster data retrieval and transaction services. *“Aside the flexibility in task definition, query languages allow the definition of complex tasks to the server better than atomic steps in procedural APIs”* (5). Query optimizers gain a lot more freedom in rephrasing the query optimally for different situations. The paper also rightly mentioned that *“application integration is much higher because one central instance is in charge of data integration and consistency”*. This buttresses our argument that with a heterogeneous database system, data integration, consistency, flexibility, retrieval and transaction services will a lot more easy.

RasDaMan (5) is one of the systems efficiently implementing raster database management. RasDaMan is implemented as a middleware running on top of a relational database system. The RasQL query language affords the capability to semantically manipulate and retrieve raster data from the database. The expressiveness of RasQL enables a wide range of signal processing, imaging, and statistical operations up to, e.g., the Fourier Transform (5)

The paper illustrated some case studies of raster database where RasDaMan was used to successfully implement Multi-dimensional raster database management system.

Multi-Dimensional Discrete (MDD) array raster database such as 2-D seamless aerial image map, a 3-D seamless map extended in the time dimension, and a 4-D database of climate simulation results were implemented in RasDaMan (5).

There are great advantages in the enhanced functionality provided by the query language approach, and in general by bringing standard database benefits such as multiuser synchronization, transaction support, and concise, explicit schema modeling to the area of raster data management. Finally, storing geo images in the database together with and vector data not only eases administration, but also enhances data consistency considerably (5).

2.2.1 Efficient Raster Data Management in Relational Databases

For an efficient and functional raster data storage that affords precised and effective geo-scientific queries in databases, raster data are stored as an array of multidimensional discrete data (MDD). Baumann in (6) proposed managing raster as multidimensional discrete data (MDD) in databases which relieves applications from many low-level but data-intensive data management tasks with the need for any specialized imaging and visualization subsystem.

Storing raster data in a database as pure byte sequence where the DBMS has no knowledge about the underlying data semantics (pixel or tile structure), execution of effective geo-scientific queries and optimization are not possible.

Existing systems related to raster data management in a database environment, such as Oracle Spatial GeoRaster or Rasdaman are designed to support storing and querying dense multi-dimensional real-valued arrays based on tiling techniques (7). We are going to be adopting in this research, the approach of MDD array storage or tile storage which has been implemented in PostGIS 2.0.

2.2.2 Large Scale Raster Database for Geo-Services

Baumann in (8) underlined the claim that databases can introduce a new quality of geo-web service on high volume multi-dimensional earth science raster data. The paper presented a conceptual model for raster data in earth science and how an efficient architecture can be derived from it. The approach is implemented in RasdaMan system. He also discussed on how the concept can be utilized in the development of OGC's geo raster services.

The paper outlined some important factors that make database raster services outperform the file-based raster services. According to the paper, we could talk about;

- Optimised tiling which makes the server fetch much less data from disk.
- Database system enables the client to send a single complex request instead of a long sequence of atomic operations.
- There is much less communication overhead when a single complex request can be made, no intermediate results have to be transferred back and forth and hence a minimum amount of data is required to answer the client's needs.

Baumann in (9) highlighted the current envisaged use of WCPS as in navigation, extraction, and server-side analysis over large multi-dimensional coverage repositories. He discussed and illustrated the importance of SQL-like request language and database approach in geo-raster services. The request language allows navigation, extraction, and ad-hoc analysis on multi-dimensional geo-scientific raster data. For example extraction tasks like retrieving of satellite image bands, performing band combination, even deriving of vegetation index maps and classification can easily and efficiently handle on the database backend through the SQL request language.

According to Peter Baumann in (9), the request language has the following advantages:

- It has a (semi-) formal semantics which combines concise syntax and semantics specification with legibility.
- The language is declarative in that there is no explicit array iteration, thereby allowing to process arrays in any cell iteration sequence, in particular based on partitioned ("tiled") storage schemes.
- Coverages are treated in a data independent way: not only are requests independent from data encoding, but also dimensions are addressed by name and not by index, thereby avoiding an artificial dimension sorting.
- WCPS queries are safe in evaluation – every request terminates after a finite number of steps (proof omitted here, but straightforward).

2.2.3 PostGIS Raster Spatial Database

Raster support is one of the new features of PostGIS 2.0 (10). The new features enable users to store georeferenced, multiband, multiresolution, with nodata value raster coverages in Postgres/Postgis spatial database. Raster coverages are stored as tables of many tiles (Multi Dimensional Discrete Data MDD). Rasters can be loaded in any format supported by GDAL and the list of raster tables is available to applications in a table named raster columns. The capability to do raster/vector analysis is our main point of attraction to this POSTGIS 2.0. This will enable the capability of heterogeneous sensor data fusion and aggregation. It also allows for raster analysis in the raster way with a set of map algebra functions working on one pixel at a time, on the neighborhood of a pixel, on two rasters, with expressions or custom user PL/pgSQL functions. All analysis takes nodata values into account unless specified. You can edit rasters pixel by pixel, many pixels at a time, using raster coordinates or georeferenced geometries. We also have the ability to convert PostGIS rasters to geometries or to any raster format supported by GDAL. With raster, topology, routing and 3D capabilities, PostGIS is becoming a complete in-the-database GIS driven with the SQL language (10). Leveraging these capabilities to develop a heterogeneous sensor database model integrating coverage and feature data for an efficient Sensor Observation Service (SOS) is of high benefit for environmental monitoring applications.

2.3 Integrating Remote and In-situ Sensor Observations

Recently there have been some studies and projects dealing with real time integration of observations from remote and in-situ sensors. The references below dealt with the real time integration of heterogeneous sensor data types for some form of environmental monitoring and management. Our main concern is that these integrations are done at the service middleware level which faces the workload of massive data retrieval from different databases. The integration is not carried out at the database backend which could leverage massive data delivery over the network.

Carlos Rueda et al. project on real-time integration of geospatial raster and point data streams (11) which was used for estimation of accurate Evapotranspiration over an extended agricultural area (California)

The system streams coverage data from Geostationary Operational Environmental Satellite (GOES) satellite and in-situ data from the California Irrigation Management Information System (CIMIS) network of weather stations. The system calculates spatially distributed daily reference Evapotranspiration and produces corresponding daily maps for the state of California. The satellite data from GOES is used within the system for evaluating the output from the interpolation methods used.

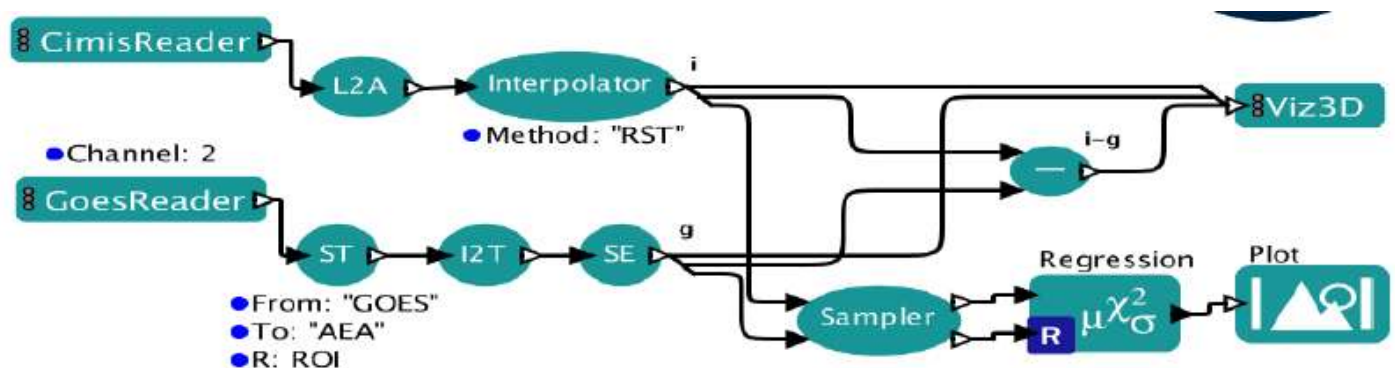


Figure 2: Workflow for comparing CIMIS and GOES temperature raster images

(Taken from (Rueda and Gertz 2008))

The resulting interpolated temperature raster is denoted as i and the GOES-derived temperature raster is denoted as g in the figure above. And the stream extension SE actor spatially aggregates the incoming images to a single composite over the covered region.

In our own case, the database backend aggregation, the fusion and aggregation of these two datasets will be seamlessly carried out at the database level and only the resulting composite images retrieved over the network. This reduces the massive workload of data retrieval from both ends as shown in the figure above.

2.3.1 Dynamic Web Mapping Service for Vegetation Productivity Using Earth Observation and in situ Sensors

Lammert Kooistra et al. worked on dynamic web mapping service for vegetation productivity, (1) integrating remote and in-situ sensor observations in a web service approach. The study is actually a sensor web based system which combines remote sensor and in-situ sensor observations to derive near real-time vegetation productivity products. The application was developed and implemented within an automated processing facility.

The approach as shown in the figure, below retrieves MODIS data from the USGS DAAC file database and the in-situ sensor meteorological data requested from the PostGIS database of KNMI SWE server through the SOS GetObservation operation. The two different data are aggregated and processed at the middleware to generate the daily GPP product. And the final mapping products are stored as ASCII raster and made available to the end user through a WMS.

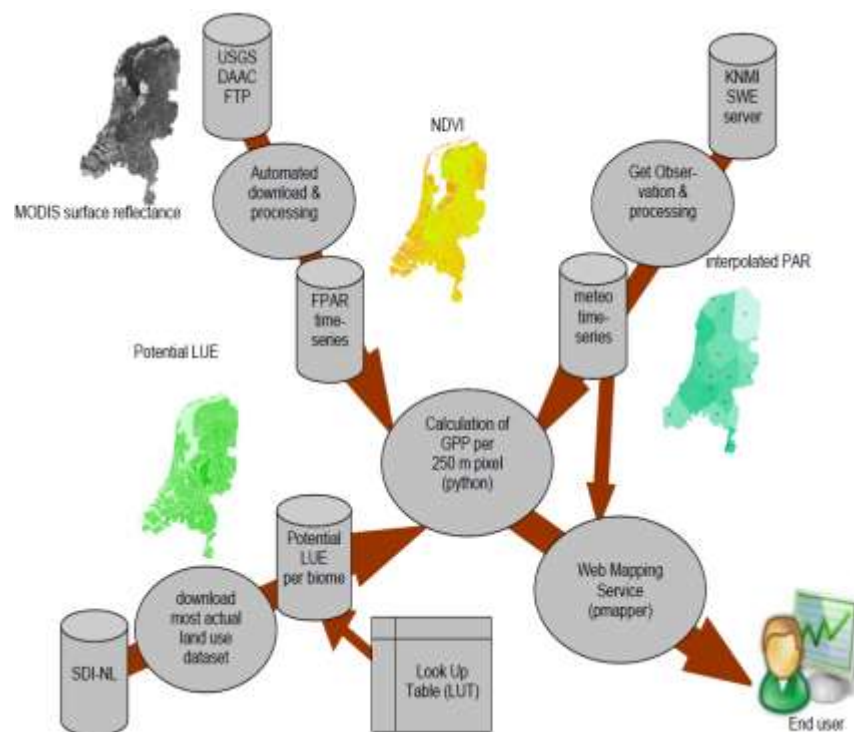


Figure 3: Overview of the communications flow and steps for the automated processing and calculation of vegetation productivity (Taken from (1))

proach whereby the remote and in-situ sensor data are stored in one database. In our approach the massive data retrieval and processing at the middleware is reduced because the different sensor data aggregation and fusion can be carried out at the database backend.

2.3.2 WARMER- Water Risk Management in Europe Project

The WARMER project presents a web-based monitoring and management system that integrates and fuses medium to high resolution ocean colour satellite remote sensing data with in-situ measurements from the buoy monitoring systems. A "production line" for daily acquisition and generation of medium resolution satellite images (MERIS FR- Medium Resolution Imaging Spectrometer - full Resolution) was developed and implemented for WARMER. For In-situ data acquisition, the ADDP (Aberdeen DISPRO DB Proxy) for acquiring and communicating the in-situ data from multiple external databases was used. These information sources were integrated in the WARMER monitoring and management system, which is an OGC (Open Geospatial Consortium) compliant system offering results in form of WMS and WFS to the users (12)

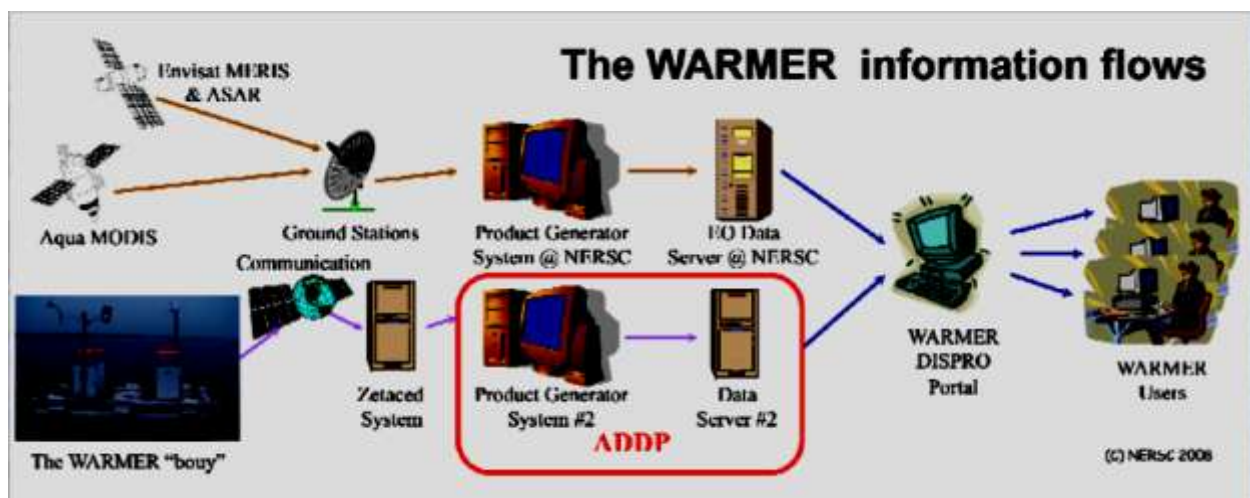


Figure 4: Image showing WARMER information flows (taken from (13))

This is also another concept of real-time integration of remote and in-situ observations on the service end after heterogeneous sensor data retrieval from different database ends over the network. Massive workload on the service end different from our approach transfers the data fusion and integration workload to the database backend.

2.3.3 Intelligent Sensorweb for Integrated Earth Sensing (ISIES) Project

The main objective of this project is to develop an intelligent sensor web system that integrates in-situ sensors with remote sensing and auxiliary data to provide improved predictions of crop

and rangeland yield. ISIES is an on-line system that integrates an in-situ sensorweb, remote sensing imagery and Geographic Information System (GIS) data to provide superior estimates and predictions of biomass, crop yield and drought severity through open and standard interfaces (14). The ISIES host server is built to retrieve data from the sensorweb, process and store it in the relational database. It is actually a host of databases built on oracle 10g and file storage system that contains all ISIES data including remote, in-situ and GIS data. Through the automatic communication SmartCore devices, the in-situ data are retrieved on daily basis. A data fusion engine was developed to automatically run the plant models that intergrate in-situ and remote sensing data and the model are run on daily basis. Yield and biomass maps are produced on daily basis for each of the test sites.

Actually this server is built around Java and database connectivity (JBDC) is Java-based. This is also the approach of multiple databases and data retrieval from different databases and processed at the server end.

Hamre (2) proposes the Integration of remote sensing, in-situ and model data in the marine information system (2) and discusses about the usefulness of integration of in-situ and remote sensor observation in the marine information system (MIS). He mentioned that the integration of these different types of data in one information system will enable combination of data from different sources so that more information can be extracted than when analyzing them separately. His emphasizes here is actually on the usefulness of integrating these different sensor data in an MIS just like it is done in a GIS to generate more useful information about the marine environment. So it is not actually on the database approach that we are proposing.

Zaks et al. (15) reviewed the components and organization of an agro ecological sensor web that integrates remote and in-situ sensor data with models to provide decision makers with effective management options at very good spatial and temporal scales. This enables the decision makers to make more informed decisions about agricultural productivity and food security. This is another paper that discussed about the usefulness of having a system that integrates remote and in-situ sensor observations for the agro-ecological sensor web. It is a

useful insight for our research but is different from our approach because the data integration is not done at the database backend.

Stonebraker et al. (16) presented a benchmark that concisely captures the database requirements of a collection of earth scientist working in the SEQUOIA 2000 project. The authors emphasized the fact that earth scientist in the project dealt with four kinds of data; raster, point, polygon and detected graph data. Therefore there was a great need to develop database benchmarks that would enable integration of these kinds of data at the database backend. This would enable effective geo-scientific queries that run some kind of aggregation across the different data types on the database. This is also in line with our proposal for heterogeneous sensor database in the Sensor Observation Service (SOS) of the SWE.

2.4 Sensor Web Enablement, SWE

Sensor Web Enablement (SWE) in the Open Geospatial Consortium, (OGC) context refers to web accessible sensor networks and archived sensor data that can be discovered, accessed and, where applicable, controlled using open standard protocols and interfaces (APIs) (17). OGC is working on harmonizing SWE standards with other OGC standards for geospatial processing standards. The SWE standards enable web-based discovery, exchange and processing of sensor observations as well as the tasking of the systems. (17)

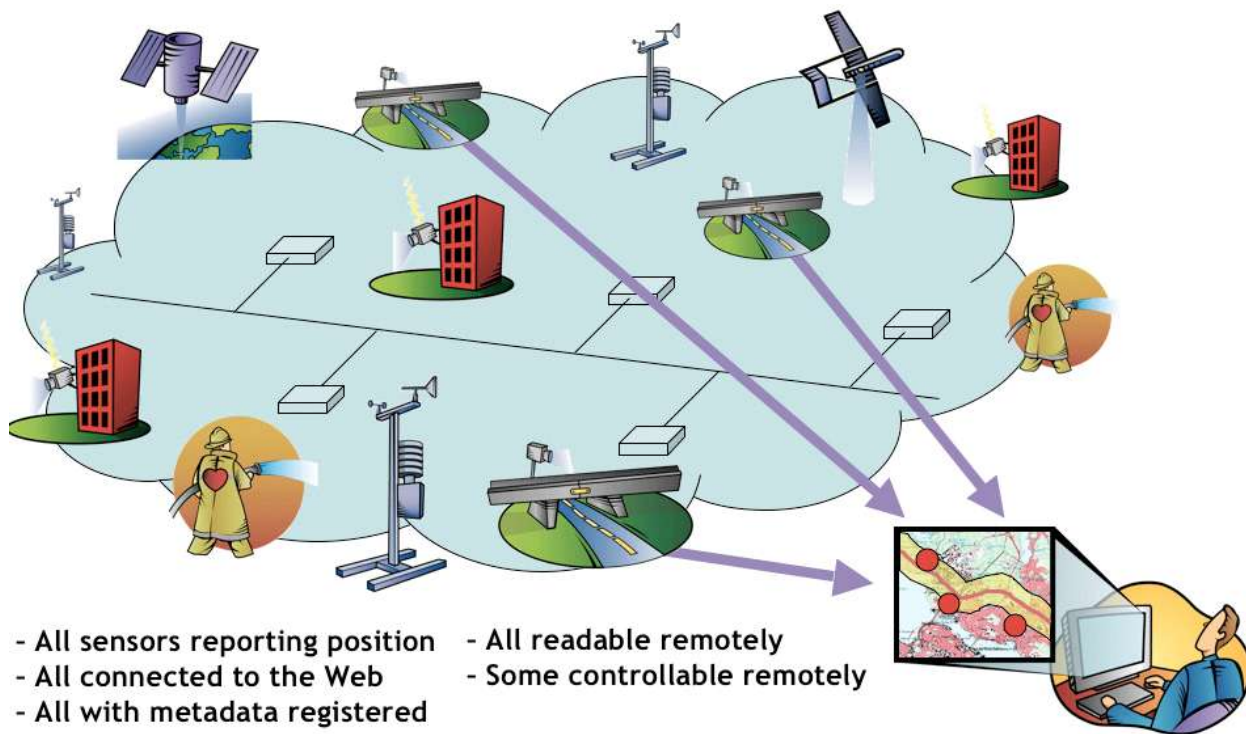


Figure 5: Sensor Web Concept

(Source of Image courtesy of the OGC)

The OpenGIS Standards that make up the SWE suite of standards includes:

The Observations & Measurements Schema (O&M), Sensor Model Language (SensorML, Transducer Markup Language (TransducerML or TML), Sensor Observations Service (SOS), Sensor Planning Service (SPS), Sensor Alert Service (SAS) and Web Notification Services (WNS). They specify the encodings for describing sensors and sensor observations and interface definitions for web services. The research area of this thesis falls on the Sensor Observation Service (SOS) domain whereby these heterogeneous sensors on the Sensor Web can be integrated on a heterogeneous database for the sensor observation service.

2.5 Sensor Observation Service SOS

The SOS is one of the main components of the SWE architecture. It defines network-centric data representations and operations for accessing and integrating observations from sensor systems. The OpenGIS Sensor Observation Service Interface Standards defines an API for managing deployed sensors and retrieving sensor observation data. (17). The SOS serves as the intermediate between the client and the observation database or near real-time sensor

channel. The SOS can also be accessed by the client to obtain metadata information about the associated sensors, platforms, procedures and other information associated with observations. Our research proposal relates to the SOS by defining database models for integration and retrieval of remote and in-situ observations at the database backend of SOS implementation.

2.6 Web Coverage Service WCS

The OGC Web Coverage Service (WCS) supports electronic retrieval of geospatial data as "coverages" – that is, digital geospatial information representing space/time-varying phenomena. (18)

The WCS 2.0 document (18) states that *“A Web Coverage Service (WCS) offers multi-dimensional coverage data for access over the Internet”*

According to the OGC WCS 2.0 (18) documents a WCS provides access to coverage data in forms that are useful for client-side rendering, as input into scientific models, and for other clients. The WCS may be compared to the OGC Web Feature Service (WFS) and the Web Map Service (WMS). As WMS and WFS service instances, a WCS allows clients to choose portions of a server's information holdings based on spatial constraints and other query criteria.

Different from the Web Map Service WMS, which portrays spatial data to return static maps (rendered as pictures by the server), the Web Coverage Service provides available data together with their detailed descriptions; defines a rich syntax for requests against these data; and returns data with its original semantics (instead of pictures) which may be interpreted, extrapolated, and even reused not just portrayed.

And also unlike the Web Feature Service WFS, which returns discrete geospatial features (vector data), the Web Coverage Service returns coverages representing space/time-varying phenomena that relate a spatio-temporal domain to a (possibly multidimensional) range of properties. As such, WCS focuses on coverages as a specialized class of features and, correspondingly, defines streamlined functionality.

WCS 2.0 uses the coverage model of the GML Application Schema for Coverages (19) which has been developed with the goal that coverages handled by a WCS can be more easily interchanged with other OGC services. WCS 2.0 supports all coverage types supported by said

Application Schema; it is not constrained to quadrilateral grid coverages like previous WCS versions (19). The WCS is of high interest for our work as it is where the problem of massive data retrieval over the network lies.

2.6 Web Coverage Processing Service WCPS

Web Coverage Processing Service is meant to provide an open-ended framework for submitting requests of extensive complexity for processing on the server-side and returning the results. The processing and returning of results is done in a systematic and formally defined ways (4). The WCPS is currently under development within the OGC. The WCPS working group initiated and led by Jacobs University Bremen (4) . WCPS includes the functionalities of WCS and extends it with expression language to form requests of unlimited complexity.

The WCPS provides the GetCapabilities, DescribeCoverage and ProcessCoverage options to the client. The GetCapabilities option just like in the WCS provides the client with an XML document describing the service and brief descriptions of the data collections from which the client may request coverages. The DescribeCoverage operation allows the client to request for a full description of one or more coverages and get an XML document description of the coverages in return. The ProcessCoverage operation provides the client with the option to process and analyse and extract information from the coverage sets stored in the server, both grid data and metadata. The ProcessCoverage request is systematically defined by the client in the processing language which supports coverage processing expressions of high complexity (4).The result of the processing is then transmitted back to the client or made available for download. The WCPS request language proposed by peter Baumann in (9) is an SQL-like language that allows the client to declaratively navigate extract and analyze multi-dimensional raster data. This is implemented based on database approach and it is advantageous because the SQL is much formalized, declarative, data independent, optimizable and safe in evaluation (9).This research concept of heterogeneous database for integrating remote and in-situ observation at the database backend is intended to leverage solutions in the WCPS which involves massive coverage data retrieval.

CHAPTER 3

3.0 REQUIREMENT ANALYSIS

In this chapter we analyse the database system requirement for seamless integration of remote and in-situ sensor observations at the database level. The analysis is done taking into consideration the varying properties and the underlying semantics of these two different sensor datasets (raster and vector). The database model for this purpose will be design as a spatial database model based on OGC standards. That is to say we treat in-situ sensor observations as time series vector coverage and remote sensor observations as also time – dependent raster coverage.

This requirement analysis study is divided into four sections:

1. Analysis on the two different datatype contents of the proposed database, differences and similarities with a view in mind of possible integration option of the two datasets considering their common geometry inheritance properties.
2. Database requirement analysis for an effective and efficient storage of time series in-situ sensor observations which enables seamless integration with raster coverage data and retrieval.
3. Database requirement analysis for an effective and efficient storage of time series and multi-band remote sensor observations which enables seamless integration with feature (vector) data and retrieval.
4. System analysis of the spatial database management system to be used based on its capabilities to handle vector and raster over other spatial databases.

3.1 Datatype Analysis; Differences and Commonalities

Looking at sensor observations from the coverage perspective, this gives rise to two types of coverages, the vector coverage and raster coverage. Coverages have some fundamental properties, exploring some of these properties and how vector and raster coverages inherit these properties, we can conceptually map out an intersection that will underline the seamless integration of the two.

ISO 19123:2005 defines a conceptual schema for the spatial characteristics of coverages. Coverages support mapping from a spatial, temporal or spatiotemporal domain to feature attribute values, where feature attribute types are common to all geographic positions within the domain. According to ISO 19123: 2005 *“a coverage domain consists of a collection of direct positions in a coordinate space that may be defined in terms of up to three spatial dimensions as well as a temporal dimension”*- (20). We have coverage examples such as rasters, triangulated irregular networks, point coverages etc.

“Coverages are like mathematical functions, they can calculate, lookup, intersect, interpolate, and return one or more values given a location and/or time. They can be defined everywhere for all values or only in certain places ” - (21). Referencing ISO 19123:2005, the area covered by a coverage is referred to as the domain of that coverage while the values that can be returned by the coverage over the area are called the range of the coverage.

The range of coverage can be numbers, nominal values (for example the name of a town given the location) as well as spatial type (vector or raster).

“Querying a coverage for a set of values given a location is called evaluating the coverage. Querying for a (set of) location(s) given some criteria on the values is given the name evaluateInverse by ISO 19123. The specifics of how answers are generated for these queries depend on the type of coverage and how the “supporting data” are stored” - (21).

A coverage is created as soon as a way to query for a certain value given a location is created. Coverages can be categorised into two, continuous and discrete coverages. Continuous coverage returns a different value of a phenomenon at every possible location within the domain. An interpolated temperature coverage which is derived from temperature data from a set of weather stations can be an example of a continuous coverage. Discrete coverages can be derived from the discretisation of a continuous surface. A discrete coverage consists of different domain and range sets. The domain set consists of either spatial or temporal geometry objects, finite in number. The range set is comprised of a finite number of attribute values each of which is associated to every direct position within any single spatiotemporal object in the domain. That is to say, the range values are constant on each spatiotemporal object in the domain.

Raster and vector coverages are both types of discrete coverage. They differ only in how they store and manage their collection of data. As coverages, they allow for basic query functions such as select, find, list etc. to be carried out on them.

Vector coverages handled as tables are the most common type of coverage implemented in most of the spatial database management systems. Individual data item are stored on each row in the table. The columns of the table ensure that collection is self-consistent. Texts are placed in text columns, numbers in numeric columns, and geometries in geometry columns and so on. The basic requirement a table must have for potential supply of information to a coverage is to have at least one geometry column and one additional column for a value or an attribute.

Raster coverages are handled as an array of multidimensional discrete data. In PostgreSQL/PostGIS precisely they are stored as regularly gridded data with the geometry of the domain as points and the range could be one or more numeric values (for example number of bands). Text values and timestamps may not be possible.

In-situ observations (vector data) are stored in tables with rows and columns in a relational manner, having one-to-one or one-to-many relationship. On the other hand remote sensor observations (raster) cannot reasonably be stored in tables but as gridded multidimensional array of data (array of points). That is to say we only have to leverage the concept of coverages to integrate the two tables in the database. The possible common column for the two datasets (tables) is the geometry column. The two datasets are collected by different sensor types in different coverage format but representing the same geographic area, it means that every point on the vector coverage has its corresponding point on the raster coverage.

Therefore the fundamental requirement from this data analysis that could enable us to integrate remote and in-situ sensor observations in a common database could be outlined as:

- storage of in-situ observations as vector point coverage and
- storage remote sensor observations as raster point (pixel) coverage.

Below in Figure 6 & 7 are the conceptual map and the UML model of the concept and management of coverages describing features, relationships, functions and how they present in the database. This coverage conceptual map is an edited extract of the model discussed in PostGIS Wiki - (21). This gives insight on how coverages can be effectively handled in spatial database for seamless integration of raster and vector coverages.

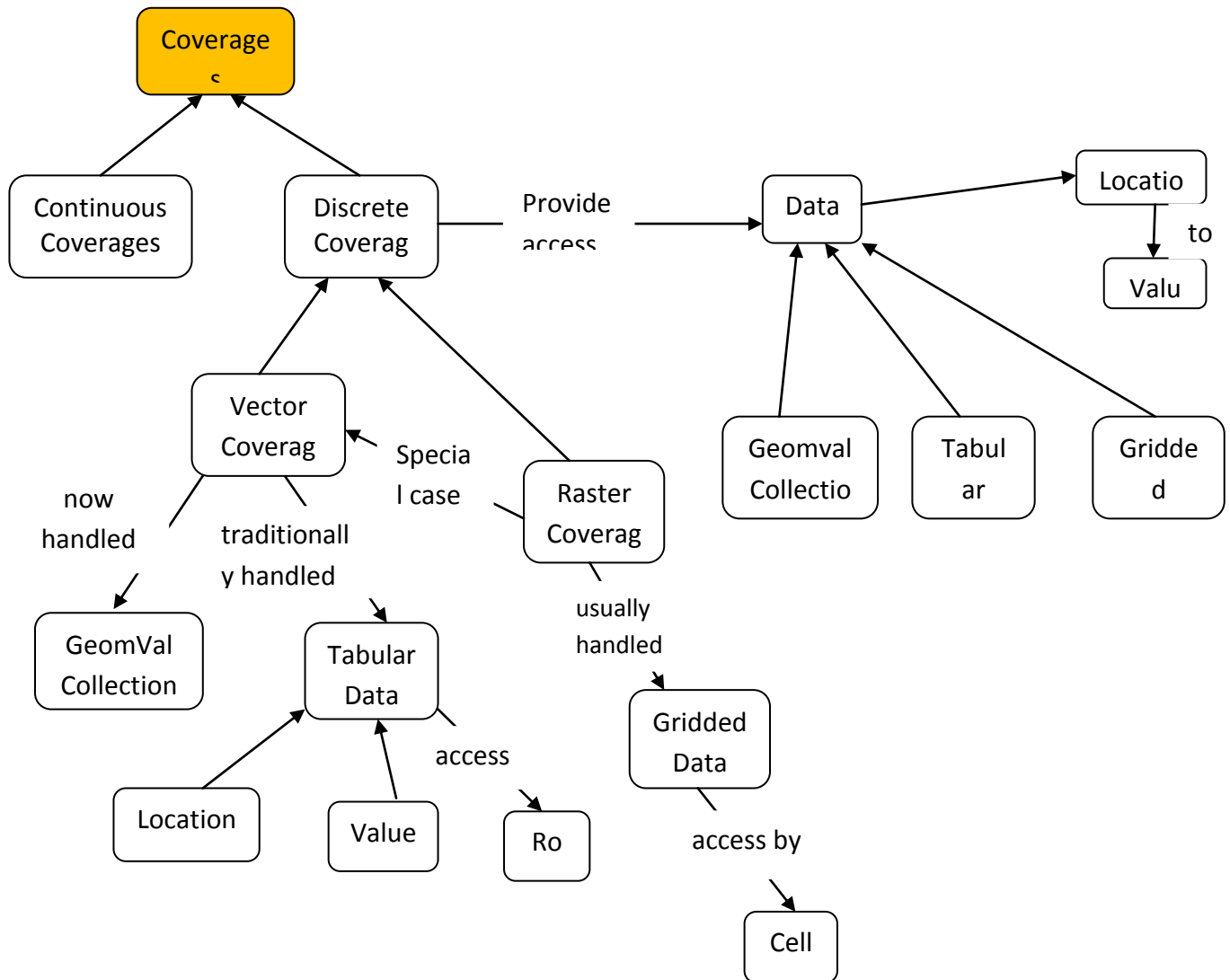


Figure 6: Conceptual Coverage Diagram

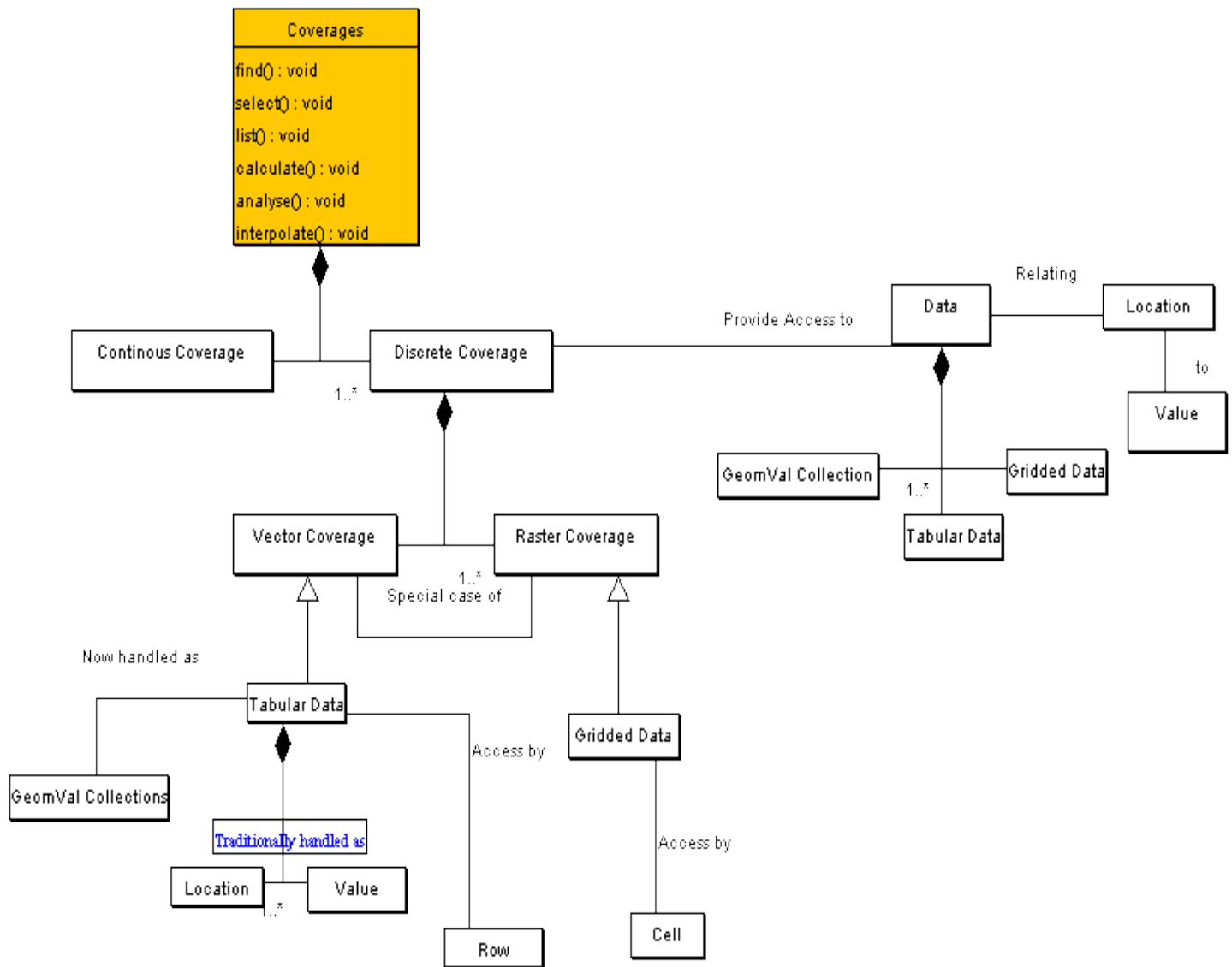


Figure 7: UML Conceptual Model of the concept and Management of Coverages

3.2 Database Requirement Analysis for In-situ Sensor Observations

Effective and efficient storage and retrieval of vector data has been well developed and implemented in most of the spatial databases such as Oracle Spatial, MySQL, Microsoft SQL Server 2008, Spatialite, Informix, Postgresql etc unlike the raster data counterpart .

But the issue here is on the requirements that can enable integration with raster data on the database as well time-effective query processing and retrieval.

2.2.1 Basic In-situ Sensor Observation Components

The basic in-situ sensor observation model is made up of five components according to the OGC Observation&Measurement specification. (22)

The components are:

- Time (observation time or sampling time)
- Location (geometry)
- Producing sensor (sensor data)
- Phenomenon which has been observed
- Observation result value (observed data)

Looking at the list above, we have all the elements to create a coverage. The location (geometry) column serves as the domain of the coverage and the rest of the columns (attributes) as the range of the coverage. We will explore the use of this geometry coverage inheritance to examine heterogeneous and homogenous geometry column integration.

We will treat in-situ sensor observations as a vector coverage backed by tables. The geometry domain is point and the range is four or more columns which could be numeric, timestamp, text etc. individual data item are stored on each row in the table. The columns of the table ensure that collection is self-consistent. Texts are placed in text columns, numbers in numeric columns, and geometries in geometry columns etc. With these items in place a vector coverage is built.

Table 1: NOAA Air temperature table sample (Taken from: (23))

ROW	DATE	LATITUDE	LONGITUDE	AIR TEMPERATURE
1	2000-08-03 00:10:45	22.8079	-163.0673	28
2	2000-08-03 00:20:45	22.8015	-163.0389	27.9
3	2000-08-03 00:30:45	22.7955	-163.0116	27.7
4	2000-08-02 08:00:44	23.5238	-165.8295	27.2
5	2000-08-02 08:10:44	23.5168	-165.8007	27
6	2000-08-02 08:20:44	23.5099	-165.7719	27
7	2000-08-02 08:30:44	23.5026	-165.7436	27

Table1 is an example of a vector point coverage. The longitude and latitude column can be used to build the geometry column which offers a point domain and the range is the columns with values. The observations will be stored as time series vector point coverage.

3.2.2 Fundamental database operation requirement for integration

Leveraging the concept of coverages, the functions and operation that can be carried out on coverages, the database can offer some of the following fundamental operations and functions on the vector coverage of in-situ sensor observation for possible integration and analysis with raster coverage.

- Intersection operation
- Buffer operation
- Overlay operation
- Interpolation operation (vector to raster conversion)

With these operations, we can easily run queries for example that can lift a point on the vector coverage, intersect it with the geometrically corresponding point or cell on the raster coverage on the database and return a value. The goal is for us to be able to do relation and overlay operations on the different coverages irrespective of how the coverages are stored. The following intersection and overlay cases can be obtainable.

- Vector to vector intersection or overlay
- Vector to raster intersection or overlay
- Raster to vector intersection or overlay
- Raster to raster intersection or overlay

The buffer operation enables the selection of a point on the vector coverage, run a buffer of any extent over it and probably use the result for a possible intersection and overlay operation with a raster coverage.

The interpolation operation offers vector to raster conversion which can be used afterwards for raster to raster intersection, overlay or map-algebra operations.

3.2.3 Spatial Indexing:

Indexes are very important in spatial databases. They are what make search in large data sets possible. Without indexing, any search for a feature would require a sequential scan of every record in the database. Indexing speeds up searching by organizing the data into a search tree which can be quickly traversed to find a particular record. (24)

There are different types of indexing supported by most of the spatial database management systems.

The B-Trees which are mainly suitable for data that are sorted along one axis, for example numbers, letters, dates etc. This type of indexing does work for geospatial data because spatial data cannot reasonably be sorted along one axis. Therefore this cannot be suitable for sensor observations.

The R-Trees are suitable and supported by some spatial databases to index spatial data as well as the GiST indexes. The most popular open source spatial database system PostgreSQL/PostGIS does not have R-Trees well implemented as it did for GiST.

GiST stands for "Generalized Search Tree" and is a generic form of indexing. GiST is used to speed up searches on all kinds of irregular data structures (integer arrays, spectral data, etc) which are not amenable to normal B-Tree indexing.

Once a GIS data table exceeds a few thousand rows, there is need to build an index to speed up spatial searches of the data unless all searches are based on attributes, in that case a normal index on the attribute field could serve.

GiST indexes have two advantages over R-Tree indexes for example in PostgreSQL. GiST indexes are "*null safe*" (24), which means, they can index columns which include null values. Secondly, GiST indexes support the concept of "*lossiness*" (24), which is important when dealing with GIS objects larger than the PostgreSQL 8K page size. *Lossiness allows PostgreSQL to store only the "important" part of an object in an index in the case of GIS objects, just the bounding box. GIS objects larger than 8K will cause R-Tree indexes to fail in the process of being built* (24).

3.3 Database Requirement Analysis for Remote Sensor Observations (Raster data)

A coverage is created in the database once there is a geometry domain (column) and a range (column with values). This goes for raster coverages, In PostgreSQL/PostGIS for example a raster coverage is created by having a geometry column called raster and attribute columns containing the attributes to the raster (e.g. band number).

The fundamental database or storage support needed on the raster coverage for efficient and seamless integration and analysis with vector coverage in the database includes the following features:

Tiled raster storage: For pixel-level query, analysis and geoprocessing, the raster coverage must be stored in the database as gridded tiles (pixel) of data. One table row represents one raster tile. Each raster has associated attributes such as pixel size, width, height, georeferencing, number of bands, nodata value etc.

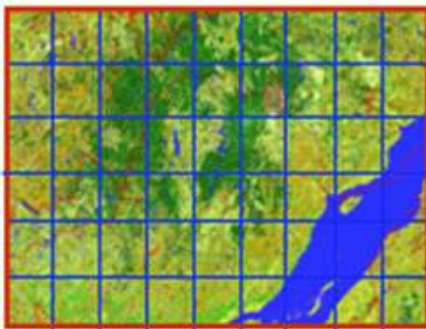


Figure 8 : Regularly tiled rectangular raster coverage

Georeferencing: The tiles must be georeferenced or geolocated in the database so that intersections and overlay operations with other data vector or raster can be possible.

Multiband and Multi-resolution support: There should be a support for multiband and multi-resolution storage. Each raster tile having bands with different pixeltypes as well as support for nodata value pixels.

Multi-Format Import and Export: Support for multi-format input and output to the database is very vital. Remote sensor observations are encoded in different formats such as geotiff, tiff, png, Jpeg, hdf, etc.

Image Pyramid and compression: Image pyramid can be used to build multiple mosaics of images, each one at a different zoom level. It helps also to speed up image handling and processing. This is very important to support data analysis at different scales and efficient storage of large size images.

Metadata Registration: Metadata of the observing sensor and observed data (raster data) should be supported in the database. Information such as name, sensor id, sensor model, spatial coverage, temporal coverage, ancillary data, quality, platform, etc. These information can be maintained in a separate metadata table outside the raster so that it can be conveniently used by applications and database optimizers outside the stored raster without processing the image.

Structured Query Language (SQL) Raster Functions and Operators Support: Raster manipulation and analysis functions and operators must be well supported by the database for raster geoprocessing and analysis at the database backend. And also the operators and functions for seamless integration of the raster with vector coverages in the database.

3.4 Spatial Database Management System Capability Analysis

Some of the existing relational databases management systems have relatively good support for vector coverages. Oracle GeoRaster and PostgreSQL/PostGIS databases only have substantial support for raster coverage management. Meanwhile Oracle GeoRaster was primarily created for raster data storage with less support for raster data analysis in the database. In view of that, presently Oracle GeoRaster does not have the full support for our proposed heterogeneous sensortype database management and analysis system. However PostgreSQL/PostGIS2.0 has relatively good raster support, functions and operations that we can leverage for the feasibility of our research goal. In addition PostgreSQL/PostGIS2.0 can be configured with python GDAL-bonded to leverage more functionality. We shall briefly discuss some the PostgreSQL/PostGIS2.0 functionalities and capabilities over Oracle GeoRaster.

PostGIS 2.0 capability to carry out seamless vector and raster data integration makes it favourable in this type of our work than Oracle GeoRaster. PostGIS2.0 can handle pixel-level raster analysis unlike Oracle GeoRaster whose content search is based on Minimum Bounding

Rectangle (MBR). MBR is the raster object in Oracle GeoRaster while Pixel is the raster object in PostGIS2.0. PostGIS uses Geospatial Data Abstraction Libraries (GDAL) to handle multi-format image input and output and when working with out-db-raster, this is a powerful functionality. To achieve robust raster geoprocessing and analysis with Oracle GeoRaster as good as it can be done in PostGIS, it requires a middleware application such as RasdaMan. RasdaMan has been implemented on top of Oracle GeoRaster for raster coverage analysis.

Table2 below is an evaluation matrix table showing the functionalities of PostGIS Raster and Oracle GeoRaster based on the necessary system requirement for full raster data management and analysis in the database.

Requirements	Oracle GeoRaster	PostGIS WKT Raster
Specific Data Type	SDO_GEORASTER	WKT Raster
Multidimensional Support	Up to 3	Up to 3
Georeferencing	Fullfilled	Fullfilled
Image pyramids	Fullfilled	Fullfilled
Partitions	Only regular	Only regular
Raster compression	Fullfilled	Fullfilled
Scan order	Not Fullfilled	Not fullfilled
Analysis capability	Not fullfilled	Fullfilled (+ r&v)
Slicing	Only get 1 layer	Only get 1 layer, planned
Subsetting	Fullfilled	Not Fullfilled (planned)
Content-based search	Using vector MBR	Partially (topological planned)
Spatial Indexing	Fullfilled (over MBR)	Fullfilled (over cells)
Open specification	Fullfilled	Fullfilled

Table 2 : Evaluation Matrix between PostGIS Raster and Oracle GeoRaster Functionalities (taken from: (25))

From this requirement analysis study, we conclude to adopt the approach of modeling the heterogeneous sensor database in the sense that in-situ and remote sensor observations are stored as coverages (vector and raster respectively) in the database. As such, we can leverage the coverage geometries for integration and analysis between the coverages. Also we tend to leverage some of the interesting PostGIS 2.0 functionalities to prototype the database schema.

CHAPTER 4

4.0 CONCEPTUAL DESIGN AND MODELLING OF THE DATABASE SCHEMA

In chapter 3 we carried out the system requirement analysis and addressed the current challenges of having a heterogeneous sensortype database that integrates remote and in-situ sensor observations. In this chapter we are looking at the possible conceptual design and model for this database. The UML model in figure 9 shows the high level abstraction model of the different classes (tables), their attributes and important operations that can be carried out on them. It shows the relationships and the logic between the classes which enable integration between these classes. The ER diagram describes the logical design or abstraction of the entities, the fields in each class and the relationships between entities. Also in this chapter we developed the conceptual model of how the database model can be integrated with other web services seamlessly. The concept of the Web Query Service WQS is also explained.

4.2 The Heterogeneous Database Schema Entity Description

List_of_Table class is the table that contains the list of all the table names in the database. Operations like GetList_of_Tables and UpdateList_of_Table can be performed on it from the user end through our proposed SQL Web Query Service WQS. The efficacy of this table is to present to the user the names and descriptions of all the tables contained in the database. A “select * from list_of_table” SQL instruction from the client end would present a table describing all the tables contained in the database. This is a kind of DescribeTables operation by the user from the client end. This should be the first operation to be carried out by any user to have a good understanding of the database before sending requests for specific data abstractions.

Coverage class holds the id and description of each coverage contained in the database. The in-situ and remote sensor observations are stored as coverages, vector and raster respectively in the database. Therefore it is necessary to have a table that presents the collections and a short description of the coverages contained in the database. For example vector or raster temperature coverage, surface elevation coverage, air pressure coverage, Normalised Difference Vegetation Index NDVI coverage and so on. The user queries this table to have a glance of what types of coverages are available in the database and the area they present.

Operations such as UpdateCoverage, selectCoverage, listCoverage etc. can be executed on it. The table has many-to-many relationship with the Feature_of_Interest, many-to-one relationship with the Observed_Phenomenon tables and one-to-many relationship with the Observation table. The table's primary key is made foreign keys in the Feature_of Interest and Observation tables. That is to say that every coverageID entry in those two related tables mentioned above must already exist in the coverage table.

4.1 UML Conceptual Schema Model of the Proposed Heterogeneous Sensor

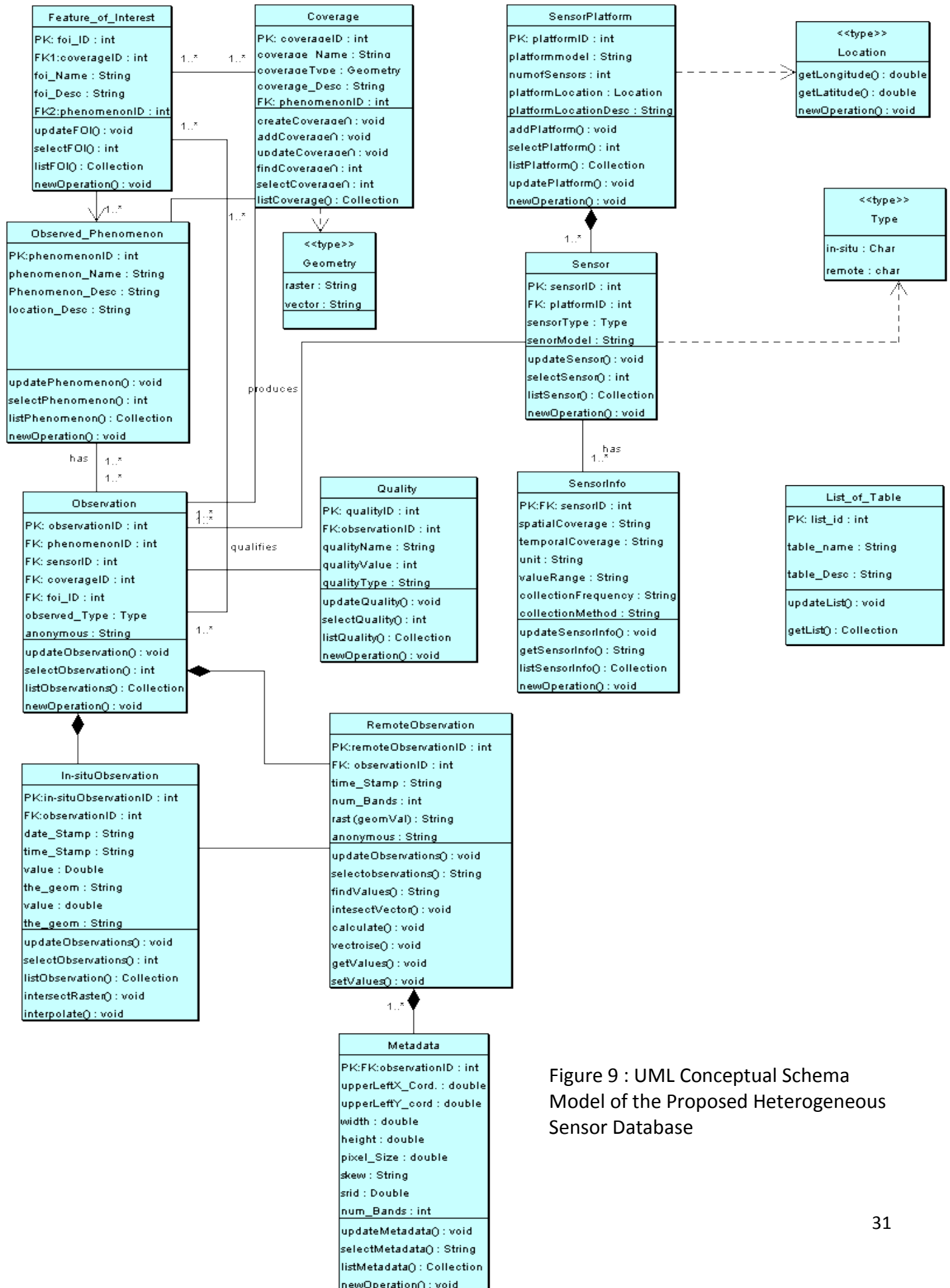


Figure 9 : UML Conceptual Schema Model of the Proposed Heterogeneous Sensor Database

The coverage class has an interface type class which defines the data type of the coverageType attribute of this class. The interface type is named Geometry, it has two values defining coverage type which can either be raster or vector coverage type.

Observed_Phenomenon class is the table that contains the names, descriptions, coverage type etc. of the various geographic phenomena that are contained in the observations. This is different from the features of interest table which contains the different features or formats of these observed phenomena that are of special interest. For example we have in our database, phenomena such as sea surface temperature, elevation, air pressure, wind speed etc., but from these phenomena, we have features such as the observations made in the night, observations during the day, observations postprocessed to a particular format and so on. These specifics are contained in the feature of interest table. The observed_phenomenon table has one-to-many relationships with Feature_of_interest, Coverage and Observation tables.

Feature_of_Interest class is the table that has the records of different features of the observed geographic phenomenon in the database. This is different from the Observed_Phenomenon table that has the record of the geographic phenomena observed from disparate sensors. The features of interest table can be updated, selected or listed by the user. This table has many-to-many relationship with the coverage, observed phenomenon and observation tables.

SensorPlatform class is the table with the record of the sensor platforms on which the sensors are mounted or housed. Attributes such as the platformId, number of sensors, platform location etc. are contained in the table. The table can be updated, selected from or listed. This table has one-to-many relationship with the sensor table. It has an interface type class named location which defines the PlatformLocation attribute data type that can be either longitude or latitude.

Sensor class is the table that contains the basic attributes about the observing sensor. Attributes such as the sensor platform, sensor type, sensor model etc. are contained in this table. These information are more about the sensor itself, detail information related to the data, coverage and methods of collection of the sensor are contained in the table named

sensorInfo which it has one-to-many relationship with. This table has an interface type table called Type which defines the sensortype attribute which is either in-situ or remote. The table can be updated, selected from, listed etc.

SensorInfo table contains information related to the sensor measurement and method. Attributes such as spatial coverage, temporal coverage, collection frequency, unit of measurement etc. can be found in this table. The table has many-to-one relationship with the sensor table. Operations such as getSensorInfo, update and list can be performed on the class.

Observation class is the table that connects the Sensor, Observed_Phenomenon, Quality, In-situObservations and RemoteObservations tables. Observation table does not contain the values and time stamps of each observed value, they are contained in the in-situ and remote observations tables. The table has a unique id for each phenomenon observation stored in the database. The Observation table has many-to-many with the feature of interest, observed_phenomenon and the coverage table. And Many-to-one relationship with the sensor and the remote_observation tables. One-to-one relationship with the In-situ_observation and the quality tables.

In-situObservation class is the table that contains the complete data of each observation that is contained in the Observation table where observationType is in-situ. It has one-to-one relationship with Observations table. The relationship between this table and the remote_observation table are handled on the fly leveraging the PostGIS intersection operation because the two coverages are handled differently in the database. Basic operations as well as complex operations such as intersection with raster, interpolation or rasterisation can be carried out on this class. The attribute called the_geom contains the geometry of each observed data.

RemoteObservation table contains the raster data of each observation that is contained in the Observation table, where observationType is remote. It has many-to-one relationship with the Observation table. Its relationship with the In-situObservation are executed on the fly through the geometry columns. Its attribute called rast contains the geometry or coordinate

information as well as the the data values (geomval). The intersection between the in-situ and remote observations tables is made possible through the intersection of the 'the_geom' and the 'rast' which is always executed on the fly. Also more complex operations such as calculate, vectorise, intersect with vector can be performed on this class.

Metadata tables houses some important header data about any raster data contained in the RemoteObservations table. It has many-to-one relationship with the RemoteObservation table. It can be updated, selected from, listed etc. from the user end through an SQL- language based request. This table is created implicitly and encapsulated in the remote_observation table and is used to describe the coverages.

4.3 ER-diagram and logical design of the database model

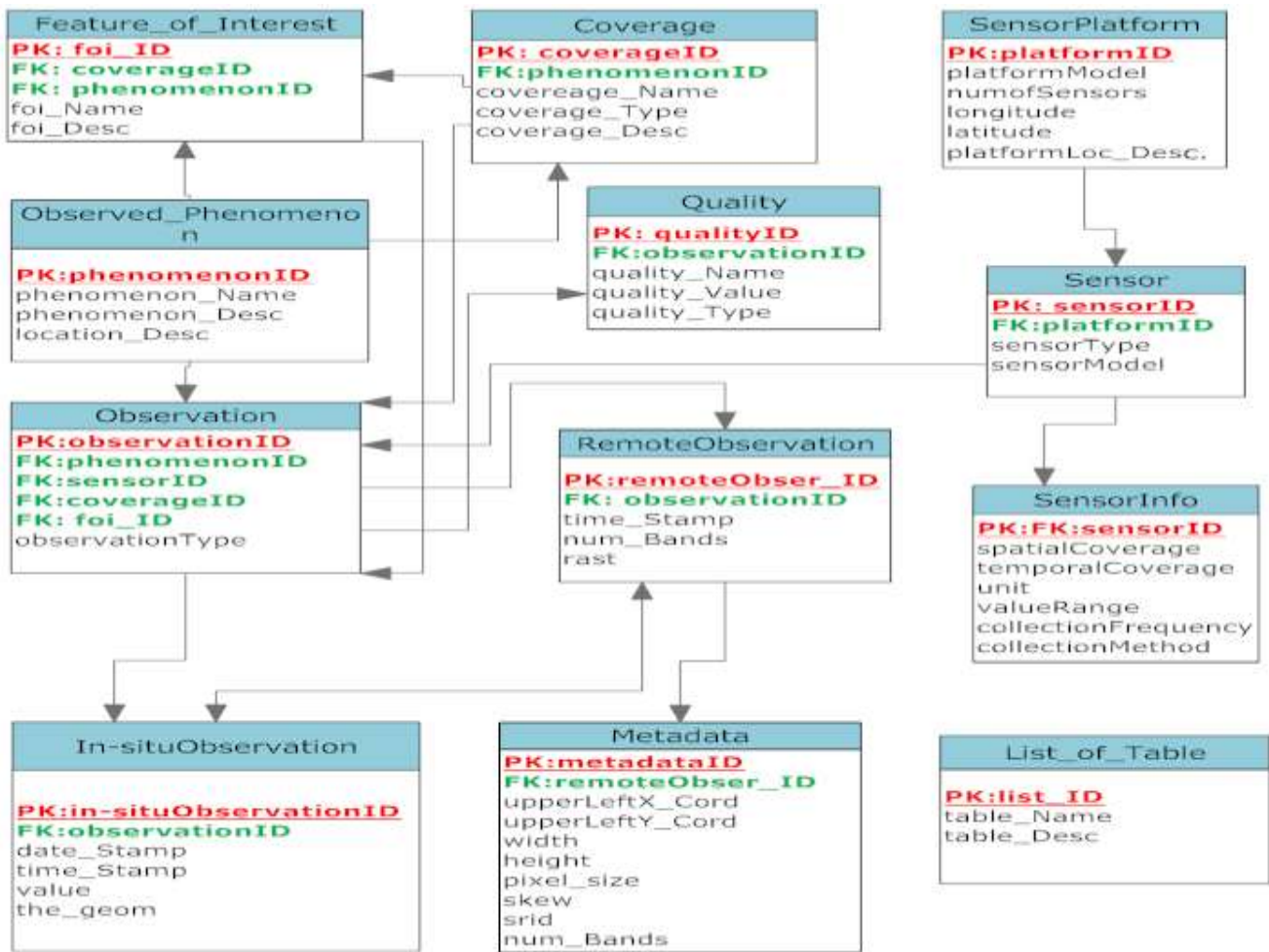


Figure 10: ER-diagram and logical design of the database model

Figure 10 is the Entity Relationship diagram and logical design of the proposed heterogeneous sensor database. The diagram shows the relationships between the tables and the attribute fields contained in each table. The primary keys PK are highlighted in red and underlined and the foreign key FK in green. In this logical design we try to avoid having intermediate tables to relate tables that have many-to-many relationship to each other. Instead we introduced foreign keys which can be leveraged by simple SQL-JOIN statements to integrate the tables.

The relationship and integration of the In-situObservation and RemoteObservation tables are executed on the fly, leveraging their geometry columns and the coverage concept that we discussed in content 3.1 above.

4.4 Integrating the Heterogeneous Sensor Database with the OGC Web Services

We are proposing an SQL-based Web Query Service WQS that delivers SQL queries from the user end to the database in the web service . This service can be intergated and accessed from within the user web or desktop application. This service provides the cleint the flexibility and ability to construct queries of extensive complexity which is delivered to the database for processing. In this case aggregations, processing and analysis of remote and in-situ observations are carried out at the database backend. The result of the query can be delivered in different formats such as ASCII, GML, KML, TIFF, JPEG etc in compliance with the OGC web mapping services, the WFS, WMS and WCS. The user specifies from the SQL query leveraging the PostGis ST_As* function, the formats the data will be delivered. ASCII or text results are delivered to the client directly from the database through the WQS. If the request result is to be delivered as a raster coverage, that means the query result is a raster or a rasterised vector and will be delivered to the client through the Web Coverage Service WCS protocol. Similar process goes for a vector or vectorised query result which is delivered through the Web Feature service WFS protocol. The request result can be delivered as a JPEG or PNG image format to the user through the Web Map Service WMS protocol.

4.4.1 The concept of the Web Query Service WQS

The Web Query Service WQS is our proposed SQL query service that serves query from the client web or desktop application to the heterogeneous sensor database. The SQL Web Query Service WQS delivers SQL queries from the client application via the web to the sensor database. It makes it easier to build and execute queries on a remote sensor database from any client application.

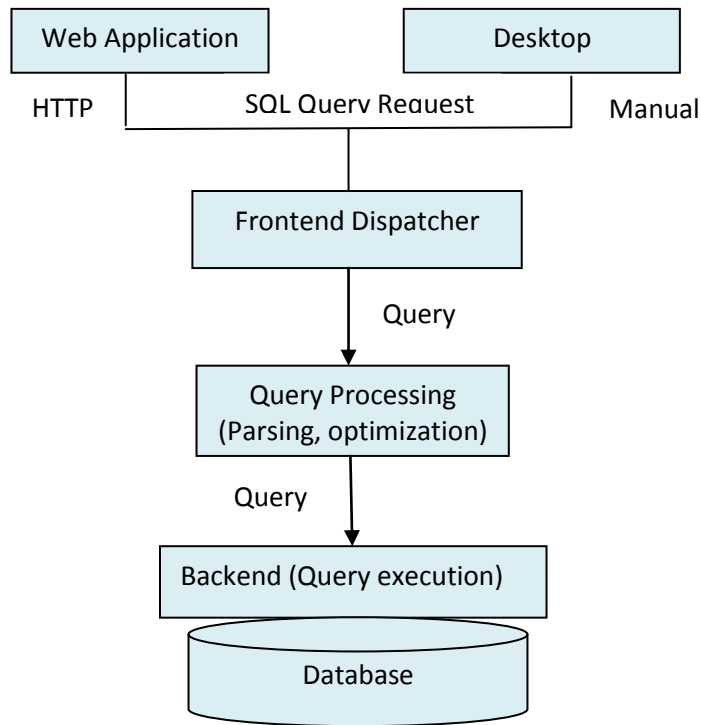


Figure 11: Conceptual Model of the proposed Web Query Service WQS

In Figure 11 the SQL query is delivered from the frontend dispatcher of client web or desktop application to the query processing and optimization module for optimization and parsing to the backend for query execution.

From a web application or API, the SQL query request is dispatched via the HTTP. From within a desktop application, a connection to the database will have to be established before queries are sent to the database for execution. Figure 12 is a sample OpenJump PostGIS database connection and query delivery interface on the client OpenJump GIS application.

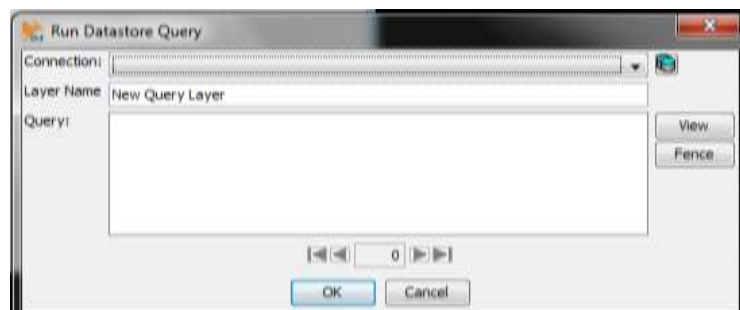


Figure 12: OpenJump Remote Database Query Connection Interface

4.4.2 Proposed Conceptual Architecture of Integrating the Heterogeneous Sensor Database and OGC Web Services.

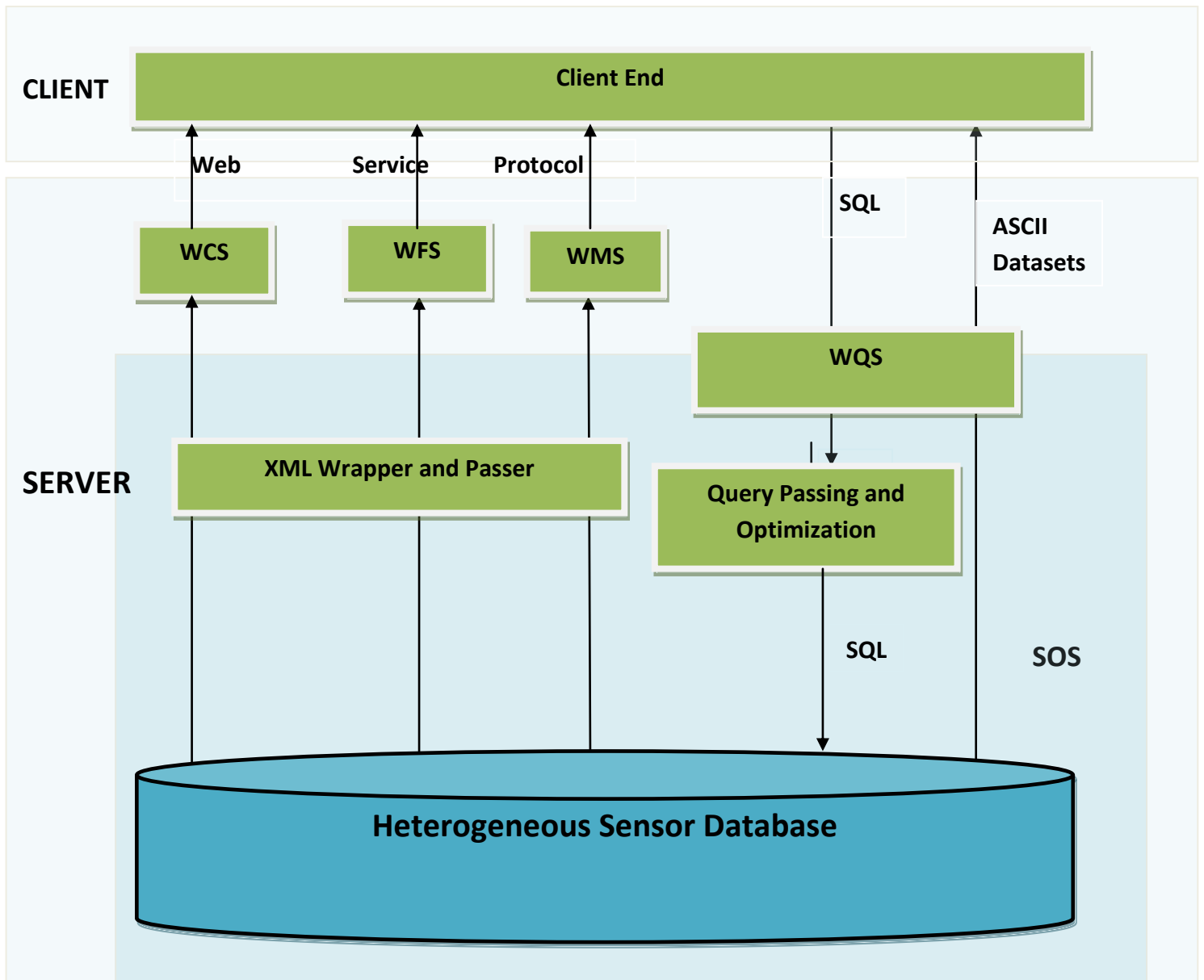


Figure 13: Proposed Conceptual Architecture of Integrating the Heterogeneous Database and the Web Services

Figure 13 describes the conceptual architecture of our proposed integration of the heterogeneous sensor database as part of the Sensor Observation Service with the proposed Web Query Service WQS and other Web Services to deliver effective results to the end user. The user on the client end, web or desktop application delivers SQL queries of any complexity through the WQS to the database. The result of the query is delivered back to the user through the relevant services depending on the format the result requested. The ST_As * PostGIS function is used in the query to specify the format of delivery. When the user specifies for example ST_As GeoTIFF, the raster coverage query result is wrapped in XML and delivered to the client through the WCS protocol. The same process goes for query results specified in ST_As JPEG, PNG and KML or GML which are delivered through the WMS and WFS respectively to the client. If no delivery format is specified in the query, the result is returned back to the client via the WQS by default in ASCII format. OGC web service operations such as GetCapabilities, DescribeSensor, DescribePlatform, GetObservation, DescribeCoverage or GetRaterMetadata, GetCoverage, ProcessCoverage etc. are carried out through this Web Query Service WQS by SQL queries. For example to describe a coverage in a raster table in the database, an SQL query such as in listing5 is delivered to the database through the WQS.

CHAPTER 5

5.0 SCENARIO DESCRIPTION

In this chapter we describe some few scenarios and use cases out of the numerous use cases where the proposed heterogeneous sensor database model can be leveraged to accomplish geo-scientific queries and processing involving remote and in-situ observations at the database end. Ranging from the a simple case where a geo-scientist would want to obtain the temperature difference between in-situ and remote temperature observations to a more complex case of estimating daily plant Evapotranspiration of a particular location. Example SQL queries are formulated and evaluated over what has been the approach on the web service.

5.1 Scenario 1: In-situ and satellite surface temperature analysis

A simple case of this scenario would be a situation where a geo-scientist is interested in determining the temperature difference between a particular in-situ temperature observation and the corresponding temperature value on the satellite surface temperature observation. Presently to carry this out on a dynamic web service, would involve the following process flow.

- Communication to the two ends of the respective databases (In-situ observation database and raster database).
- Data retrieval from the two respective databases
- Analysis and geo-processing at the service middleware
- Return of result to the client side.

In the heterogeneous sensor database approach, it only involves a direct communication between the client and the database. The communication delay to the two database and massive data retrieval and workload on the service middleware are substantially reduced as depicted in the figures 14 and 15.

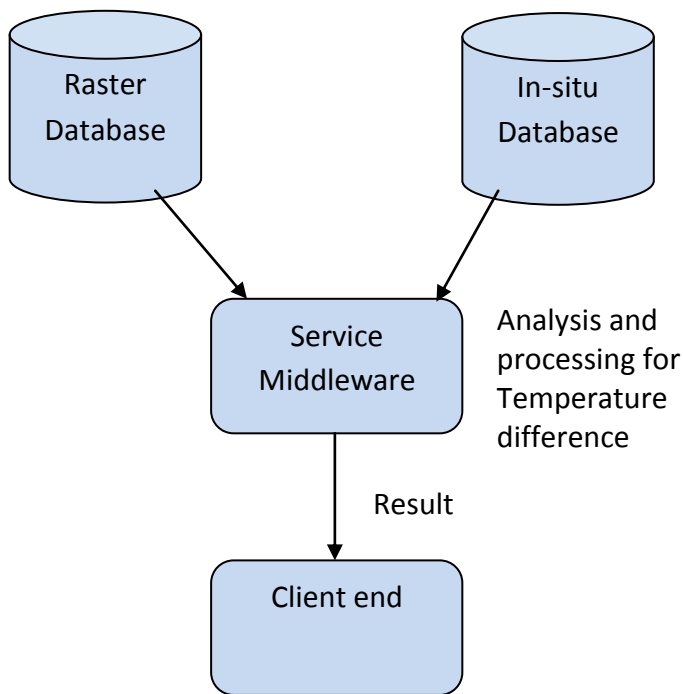


Figure 14: Multiple Database Approach

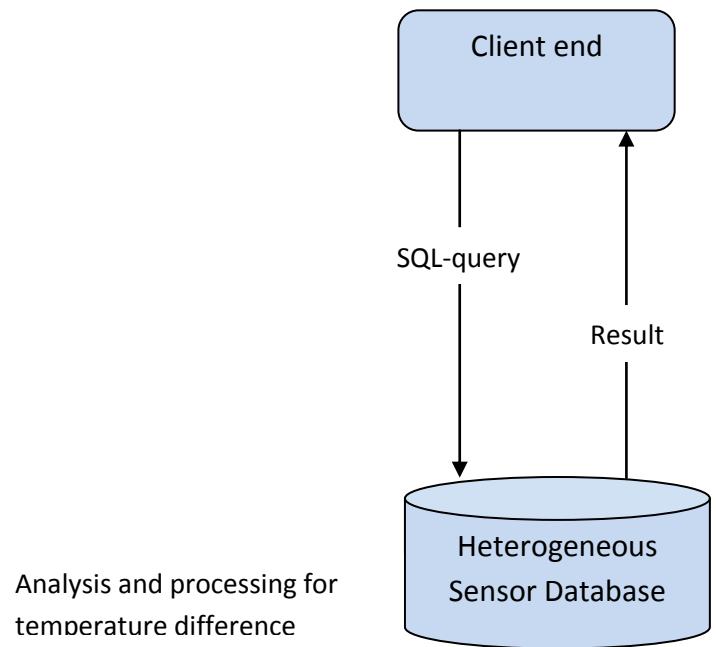


Figure 15: Common Database Approach

The heterogeneous sensor database approach gives the user on the client side the leverage to send an SQL-query request in the form in listing1 to the database and gets the required result.

Listing 1: SQL sample query for scenario 1

```

SELECT
    I_t, (tv).val as R_t, (I_t - R_t) as D_t
FROM (SELECT ST_intersection(R.rast, I.the_geom) AS tv,
        I.temp_value as I_t, ST_Value(R.rast, I.the_geom) as R_t
FROM In_situ_temp I, Remote_temp R
WHERE I.the_geom && R.rast
AND ST_intersects(R.rast,I.the_geom)
AND I.temp_id = 111111111
) foo;
  
```

The sample SQL query in listing1 above does the required analysis and aggregation of the temperature values from in-situ observation temperature table (In-situ_temp) and remote sensor observation surface temperature table (Remote_temp), I_t and R_t respectively and returns the difference, D_t. The Intersection operation (ST_intersection) is used to intersect the point geometry of the particular in-situ temperature observation (the_geom) where id is '11111111' with the corresponding geometry of the temperature pixel (rast) on the remote temperature observation.

Note: the id value '11111111' is arbitrary in these sample queries; it can become any value in the real implementation exercise.

The 'foo' is an arbitrary alias name for the sub-query in the query statement.

5.1.1 Scenario 2.1: Weighted Mean surface temperature values from vector buffers

We now describe a sample case whereby a geo-scientist would want to create buffers around the in-situ temperature observation points on the in-situ temperature observation tables, overlaps these buffers on the raster surface temperature observations and finally retrieves the weighted mean raster surface temperature values of these buffers in comma separated values CSV format.

The listing 2 sample SQL query request can be delivered to the heterogeneous sensor database through the proposed Web Query Service WQS from the client end to extract the pixel-area weighted mean temperature values of those buffers.

Listing 2: SQL sample query for Scenario 2.1

```
SELECT sum(ST_Area(the_geom)*val)/(sum(ST_Area(the_geom))) AS Meantemp
FROM (Select(ST_Intersection(R.rast, (ST_Buffer(I.the_geom,1000))))).geom AS the_geom
      (ST_Intersection(R.rast, (ST_Buffer(I.the_geom, 1000))))).val AS val
FROM In-situ_temp I, Remote_temp R
WHERE the_geom && R.rast
      ST_Intersects(R.rast, I.the_geom)
AND I.id = 111111111111
```

)foo;

Weighted mean temperature here means that the weight to each temperature pixel value is proportional to the area it occupies in the raster. For example if the area occupied by pixels with temperature equal to 22 degree Celsius is greater than the area occupy pixels with an temperature equal to 23, then 22 has a higher weight in the equation (proportional to its relative area).

5.1.2 Scenario 2.2: Day and Night temperature Difference of a Location from Satellite Surface Temperature Observations.

This is a case where one may want to obtain the night and day temperature difference of a point whose location is given in the in-situ observation table or location provided from the user end.

Listing 3 sample SQL queries delivered from the client end would deliver the required result.

SELECT

DT, NT, DT-NT **as** TD, the_geom

FROM (SELECT

ST_Value(R1.rast,I.the_geom) AS DT,

ST_Value(R2.rast,I.the_geom) AS NT,

ST_AsBinary(I.the_geom) as the_geom

FROM In-situ_temp I, Remote_Daytemp R1, Remote_Nighttemp R2

WHERE I._id = 11111111

AND ST_Value(R1.rast,I.the_geom) IS NOT NULL

AND ST_Value(R2.rast,I.the_geom) IS NOT NULL

) foo;

OR

Listing 3: SQL sample query for sample SQL queries Scenario 2.2

SELECT

ST DT, NT, DT-NT as TD, the_geom

FROM (SELECT

ST_Value(R1.rast,ST_Point(long,lat)) AS DT,

ST_Value(R2.rast, ST_Point(long,lat)) AS NT,

ST_AsBinary(ST_Point(long,lat)) as the_geom

FROM In-situ_temp I, Remote_Daytemp R1, Remote_Nighttemp R2

WHERE ST_Value(R1.rast,ST_Point(long,lat)) IS NOT NULL

AND ST_Value(R2.rast,ST_Point(long,lat)) IS NOT NULL

) foo;

This is the case where the point location interest is not already in the database but provided from the user end.

5.2 Scenario 3: Geo-scientific Analysis of In-situ Temperature /Air Pressure Data and Satellite Elevation/ Height Observation

Considering the relationship between temperature or air pressure and surface elevation, analysis between these different phenomena coming from in-situ sensor and remote sensors can be carried out conveniently in the proposed heterogeneous sensor database. For instance an earth scientist interested in ascertaining the terrain height information of a location in view of the air pressure data of this location.

In this case, the interest is to extract the height value of a particular pixel location on the satellite elevation data (e.g. SRTM data) whose air pressure or temperature value is obtained from in-situ observation at this location.

To carry out this operation on the web service with the current approach of in-situ and remote sensor observations in web services and data analysis and aggregation on the service middleware would entail the following processes as depicted in figure12a.

- Back and forth communications to the two ends of the respective databases (In-situ air pressure/temperature database and SRTM raster database).
- Massive data retrieval from the two respective databases, especially from the raster database.
- Data analysis and geo-processing at the service- middleware to extract the elevation value of the particular pixel in question.
- Return of result to the client side

This approach has the following drawbacks:

- communication delay
- massive data retrieval load
- massive processing work load on the service end.
- users don't have query request flexibility.

But in our proposed heterogeneous sensor database approach, an SQL-based instruction such as the listing 4 is delivered to the database through the proposed WQS. Data retrieval, geometry intersection and data extraction operation are carried out on the database and the result delivered back to the user in real time. This reduces communication delay, massive data retrieval and excessive work load on the web service.

Listing 4: SQL sample query for Scenario 3

SELECT

Pval , (pv).val AS Elevation

FROM (SELECT I.value, as Pval, ST_intersection(R.rast, I.the_geom) AS pv

FROM In-situ_air_pressure I, Remote_srtm R

WHERE I.the_geom && R.rast

```
AND ST_intersects(R.rast,I.the_geom)
```

```
AND I_id = 111111111111
```

```
) foo;
```

OR

In the case where more than one point is being ascertained at the same time, then we could have such like this below;

```
SELECT
```

```
    Pval , (pv).val AS Elevation
```

```
FROM (SELECT I.value, as Pval, ST_intersection(R.rast, I.the_geom) AS pv
```

```
FROM In-situ_air_pressure I, Remote_srtm R
```

```
WHERE I.the_geom && R.rast
```

```
AND ST_intersects(R.rast,I.the_geom)
```

```
AND I.value < '1111111mBar'
```

```
ORDER BY I.id;
```

5.3 Scenario 4: Estimation of Actual Crop Evapotranspiration ET

Estimation of actual Evapotranspiration according to (26), involves the integration of Normalised Difference Vegetation Index NDVI derived from satellite sensor observations and gridded reference Evapotranspiration derived from observations from automatic weather stations.

$$AET = FVC * RE, (26)$$

$$FVC = N^2$$

$$N = (NDVI_p - NDVI_{min}) / (NDVI_{max} - NDVI_{min})$$

Where: AET = Actual Evapotranspiration

FVC = Fraction Vegetation Cover

RET = Reference Evapotranspiration obtained from in-situ observation

NDVI_p = the NDVI Value at a point p

NDVI_{max} = the maximum NDVI value within the entire area of observation

NDVI_{min} = the minimum NDVI value within the entire area of observation

Now, we describe a scenario whereby a crop scientist may be interested in obtaining the actual Evapotranspiration of a location having the reference Evapotranspiration of this point obtained from the weather station on that location.

In a case like this where the interest could be only a single point location, accomplishing this on the web service with the current multiple database approach is not optimal. The massive NDVI raster image of the area or the subset would have to be retrieved from the raster database. Also the reference Evapotranspiration from the in-situ weather station retrieved from the in-situ observation database. The actual Evapotranspiration calculated on the relevant web processing service and finally the result delivered to the client. Figure16 describes the follow diagrammatically;

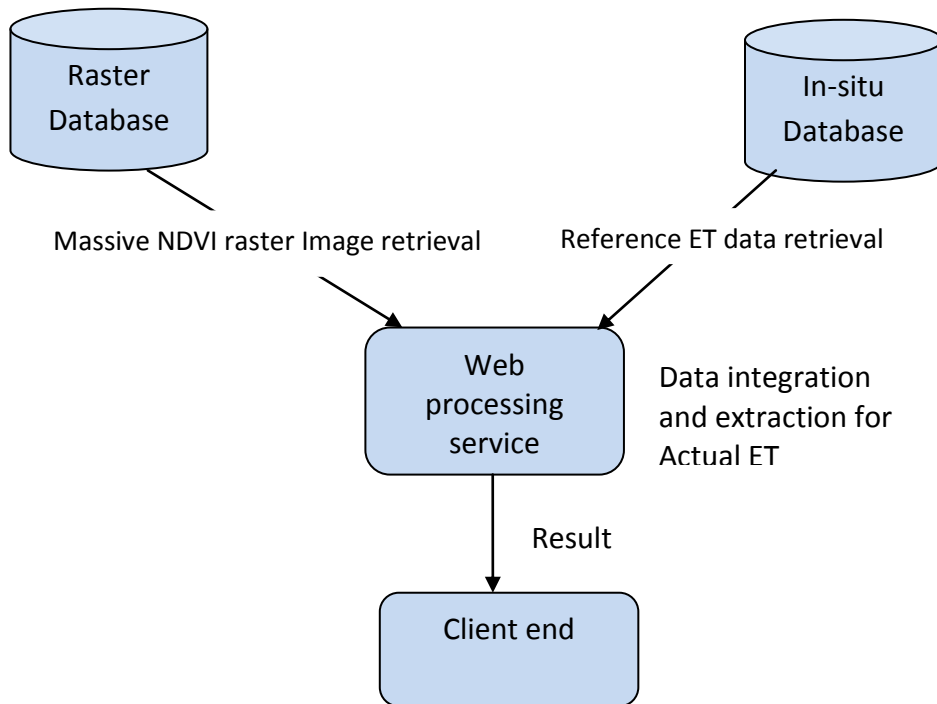


Figure 16: Multiple Database Approach for scenario 3

This double communication and long process workload can effectively be accomplished with an SQL query from the client to the proposed heterogeneous sensor database with very extensive flexibility.

When we have the NDVI raster and reference Evapotranspiration data from in-situ weather stations efficiently stored in a single database, we can leverage an SQL query like in the listing 5 below to extract an estimated actual crop Evapotranspiration at a point of interest.

Listing 5: SQL sample query for Scenario 4

SELECT

RET, NDVIp, FVC, AET

FROM (**SELECT** (pow(((NDVIp-NDVImax/(NDVIp-NDVImin)),2)) **AS** FVC,
pow(((NDVIp-NDVImax/(NDVIp-NDVImin)),2)) *RET **AS** AET,

ST_Value(R.rast,I.the_geom) **AS** NDVIp, I.value **AS** RET

FROM In-situ_RET I, Remote_NDVI R

WHERE I.id = 1111111111

AND ST_Value(R.rast,I.the_geom) IS NOT NULL)

) foo;

Note: For a less précised estimation, we can assume NDVImax = 1 and NDVImin = 0, which are the maximum and minimum NDVI values we can have belonging to thick vegetation and water surface respectively.

But for a more précised estimation, we can obtain the maximum and minimum NDVI values within our coverage area by querying the NDVI raster coverage statistics for maximum and minimum pixel values. The following SQL sample codes in listings 6 can be used in POSTGIS database to abstract these values to be used in the subsequent AET estimation.

Listing 6: SQL sample query for abstracting maximum and minimum pixel values

```
SELECT (stats).max  
FROM (SELECT ST_SummaryStats(rast) AS stats  
      FROM ndvi  
      ORDER BY stats DESC  
      LIMIT 1) AS foo;
```

```
SELECT (stats).min  
FROM (SELECT ST_SummaryStats(rast) AS stats  
      FROM ndvi  
      ORDER BY stats ASC  
      LIMIT 1) AS foo;
```

5.4 Scenario 5: DescribeCoverage (Raster Coverage Metadata) Query Operation.

The DescribeCoverage or GetRasterMetadata operation is one of the important operations provided by the web coverage service which allows the client to request for a full description of one or more raster coverages in the database. This operation can also be done by an SQL query request on the proposed Heterogeneous sensor database.

In this scenario we present a sample SQL query statement a user on the client end can leverage to describe any raster coverage of interest available on the database.

This will describe the properties of the raster table such as upper left corner coordinates, width and height, pixel sizes, skews - or rotations, SRID, number of band, pixel type, has nodata value, nodata value, etc.

Listing 7: SQL sample query for describing raster coverage in the database

```
SELECT (md).*, (bmd).*
      FROM (SELECT ST_Metadata(R.rast) AS md,
                ST_BandMetadata(R.rast) AS bmd
      FROM Remote_Temp R
      LIMIT 1
    ) foo;
```

We limit the result to one that is for the first row (tile) because the metadata for other rows or tiles are the same except for the upperleftX and upperleftY.

In general the results of the sample queries shown above for the mentioned scenarios are alphanumeric or CSV formatted. They are returned to the client directly from the database. Other results formats are also possible depending on how the client wants the results delivered. For example if the result is raster then it can be delivered as JPEG image, PNG or TIFF image formats depending on what format is expressed on the query request. In this case raster output functions such as ST_ASJPEG, ST_ASPNG, and ST_ASTIFF etc. are used in the query request to indicate the return format. The result can be delivered to client through the WCS and WMS protocol as the case may be.

Also if the query result is wanted in form of vector geometry, functions such as ST_ASGML, ST_ASKML etc. can be used in the query statement. The result can be delivered to the client through the WFS protocol.

CHAPTER 6

6.0 POSTGIS-BASED PROTOTYPICAL IMPLEMENTATION, PERFORMANCE AND HYPOTHESIS EVALUATION

Amongst the existing spatial database management systems, PostGIS 2.0, extension of PostgreSQL is the only one currently that has the functionalities for seamless integration and processing of vector and raster observations on the database. Oracle GeoRaster presently has less support for raster pixel-level analysis. PostGIS 2.0 is a recent development of the PostGIS which incorporates vector and raster data manipulation functionalities. PostGIS Raster introduces a new PostgreSQL data type called raster that stores raster data in a binary format similar to how the PostGIS geometry and geography types store vector data (27). The efficient ability of PostGIS 2.0 to handle raster data proves it advantageous for this heterogeneous sensor database model in the sense that:

- It supports multiband, nodata value, georeference, overviews, overlapping tiles and non rectangular coverages.
- It is not really limited in size (PostgreSQL has a limit of 32 TB).
- It is very well integrated with the existing PostGIS geometry type enabling seamless and efficient intersections operations with vector tables.
- It comes with a very versatile Python raster loader which supports batch loading through wildcards and as many input formats as GDAL provides (28).

Based on this factor, we decided to leverage these new functionalities of PostGIS 2.0 to do a prototypical implementation of our heterogeneous sensor database model for the Sensor Observation Service SOS. In PostGIS 2.0 we could store in-situ sensor observations as vector coverages and remote sensor observations as raster coverages, hence we could easily run extensive integration and analysis between these two coverages with the functionalities it provides.

6.1 System Configuration

For the purpose of this prototypical implementation exercise, the following software packages and libraries are needed in combination with PostgreSQL/PostGIS2.0 to realize the

functionalities required for efficient implementation of the heterogeneous sensor database model.

There are different versions of these packages not all of them works for this purpose but we had no problem with these ones:

- Python 2.7
- Numpy 1.5.1
- GDAL 1.8.0
- PostgreSQL-9.0.4.1
- PostGIS-pg-binaries-2.0.0svn
- And OpenJump-bin-1.4.2

PostGIS Raster loader makes use of GDAL Python bindings and Numpy python library for loading raster. Therefore python, numpy and GDAL are required to be installed for this purpose.

GDAL- Geospatial Data Abstraction Library is a translator library for raster geospatial data formats. It presents a single abstract data model to the calling applications for all supported formats. It also comes with a variety of useful command line utilities for data translation and processing. OGR is a subset of GDAL which in a similar way as GDAL handles simple features vector data (29) . The most interesting capability of GDAL which we are leveraging for this research is its vast support for many raster data formats. That is to say that we can conveniently have majority of the remote sensor observations of different formats loaded into the database.

Numpy is a package very vital for scientific computing with python. It affords among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities (30).

NumPy is used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of

databases. This is a very important package for this research as we leverage its capabilities to handle multi-dimensional arrays of discrete data (raster data) in the database.

OpenJump is a java open source GIS application, very easy to deploy and works very great with PostGIS. We use it in this exercise to display and visualise the geometries of the observations loaded in the database and to act as a client side desktop application to query the database.

6.1.1 Installation Procedure:

The packages were installed in the following sequential order:

- Python 2.7 was downloaded from <http://www.python.org/download/releases/2.7.1/> and installed first of all.
- Followed by numpy-1.5.1, downloaded <http://www.lfd.uci.edu/~gohlke/pythonlibs/> and installed. During installation, numpy automatically binds itself with the installed python program because it is a python package.
- GDAL installation follows, we download GDAL 1.8.0 from <http://vbkto.dyndns.org/sdk/> and installed. After installing GDAL, it has to be configured in the system to get bonded in the python libraries. This was done by adding this path 'C:\Program Files\GDAL\' in the system's environment variables and *creating the following new variables*:
GDAL_DATA=C:\Program Files\GDAL\gdal-data, GDAL_DRIVER_PATH=C:\Program Files\GDAL\gdalplugins and PROJ_LIB=C:\Program Files\GDAL\projlib.
- PostgreSQL-9.0.4.1 was downloaded from <http://www.enterprisedb.com/products-services-training/pgdownload#windows> and installed. During installation, the message asking to install other libraries using the Application Stack Builder should be cancelled or rejected because it would automatically install PostGIS 1.5.x which is different from PostGIS2.0.
- After installing PostgreSQL, we then downloaded PostGIS2.0 from <http://postgis.refractory.net/download/windows/experimental.php> and installed manually to bind with the installed PostgreSQL.
- Finally we downloaded the OpenJump software, <http://www.openjump.org/> and installed which would be used afterwards as a client side desktop application to visualise the sensor observations.

For more information on how to install and configure these packages for a running PostGIS 2.0 database management system see (31).

6.2 Test Data Collection and Description

For the purpose of this prototypical implementation of heterogeneous sensor database model and to test some of the scenarios described in chapter 4, various in-situ and remote sensor data were gathered from different free source sensor data services on the web. The data we used for this exercise were collected from the following sources.

6.2.1 In-situ Sensor Data

- Sea Surface Temperature SST, shipboard sensor observations acquired from the National Oceanographic Data Center NODC, <http://www.nodc.noaa.gov/ssd/access.html> within the Gulf of Mexico.
- Land Surface Temperature LST sensor observations acquired from the U.S. Climate Reference Network of the National Oceanographic and Atmospheric Administration NOAA, <http://gis.ncdc.noaa.gov/map/crn/>. The sensors were located within the Colorado area.
- Reference Evapotranspiration data was acquired from 12 Automated Weather Data Network (AWDN) stations within southeastern part of Nebraska where majority of landcover is agriculture. The weather stations are being operated by High Plains Regional Climate Center (HPRCC). The data was downloaded from <http://www.hprcc.unl.edu/>.

6.2.2 Remote Sensor Data

- Land Surface Temperature, LST (day and night), Sea Surface Temperature SST and NDVI remote sensor observations were acquired from NASA Earth Observations NEO, <http://neo.sci.gsfc.nasa.gov/> in GeoTiff format. The data were acquired by Terra/MODIS satellite and processed by NEO.
- Shuttle Radar Topographic Mission (SRTM) elevation data was obtained from the FTP server of Global Land Cover Facilities GLCF, <http://glcf.umiacs.umd.edu/data/srtm/>.

- Reference Evapotranspiration

The table and figures below are the visualization of some of the extracted data collected from the sources mentioned above and used for this prototypical implementation of the database model. We actually subset some fraction of the images corresponding to the area we have the in-situ observations collected.

Table 3: In-Situ Shipboard Sensor Air and Water Temperature Data

ROW	DATE	LATITUDE	LONGITUDE	AIR TEMPERATURE	WATER TEMPERATURE
1	2006-08-01 00:00:25	27.84317	-115.65133		22.747
2	2006-08-01 00:08:07	26.86673	-88.62834	29.66	30.201
3	2006-08-01 00:10:25	27.8165	-115.64017		22.740
4	2006-08-01 00:18:07	26.88345	-88.66021	29.68	30.162
5	2006-08-01 00:20:25	27.8105	-115.64		22.738
6	2006-08-01 00:28:07	26.90063	-88.69293	29.56	30.186
7	2006-08-01 00:30:25	27.8075	-115.63583		22.719
8	2006-08-01 00:38:07	26.92037	-88.72312	29.51	30.189
9	2006-08-01 00:40:25	27.7795	-115.63		22.742
10	2006-08-01 00:48:07	26.94624	-88.74693	29.48	30.261
11	2006-08-01 00:50:25	27.76667	-115.64367		22.693
12	2006-08-01 00:58:07	26.97262	-88.77004	29.53	30.205
13	2006-08-01 01:00:25	27.74567	-115.63733		22.697
14	2006-08-01 01:08:07	26.99893	-88.79314	29.41	30.210
15	2006-08-01 01:10:25	27.71767	-115.627		22.610
16	2006-08-01 01:18:07	27.0252	-88.81605	29.58	30.176
17	2006-08-01 01:20:25	27.68967	-115.61583		22.950
18	2006-08-01 01:28:07	27.04769	-88.84337	29.51	30.148
19	2006-08-01 01:30:25	27.675	-115.59567		23.240
20	2006-08-01 01:38:07	27.06974	-88.87159	29.46	30.296
21	2006-08-01 01:40:25	27.67267	-115.5745		23.275
22	2006-08-01 01:48:07	27.0918	-88.89968	29.43	30.351

Table 4: In-Situ Sensor Land Surface Temperature LST Data, Daily Average

Day	Temperature								Precipitation			Wind		Solar Radiation						
	Average		Maximum		Minimum		(Max-Min)/2	Total	Max Hourly	Max 5m	Max Hourly	Max 10c	Total (MJ/m2)	Max						
	°F	°C	°F	°C	°F	°C	°F	in mm	in mm	in mm	Mph	Mph		mmol	°F					
1	61.6	16.5	72.6	22.5	52.9	11.6	59.0	62.7	17.0	0.00	0.00	0.00	0.0	2.6	5.62	6.1	13.67			
2	58.9	15.5	73.8	22.8	51.30	53.2	51.8	34.0	83.1	17.3	0.02	0.6	0.02	6.6	0.81	0.3	2.1	4.72	4.1	9.13
3	61.5	16.4	78.4	25.8	53.95	51.6	50.9	5.00	65.0	18.4	0.09	2.2	0.04	1.0	0.02	0.4	2.9	6.46	5.9	13.11
4	60.5	15.9	76.1	24.5	52.2	11.2	58.0	64.1	17.8	0.00	0.00	0.00	0.0	2.7	5.99	6.5	14.63			
5	64.3	17.9	77.5	25.3	51.6	10.9	59.0	64.0	18.1	0.00	0.00	0.00	0.0	2.9	6.58	5.5	12.36			
6	66.5	19.2	80.1	26.7	53.55	52.0	52.0	66.8	19.4	0.00	0.00	0.00	0.0	3.0	7.29	6.4	14.43			
7	66.4	19.2	80.4	26.9	58.9	15.5	59.0	65.8	18.8	0.00	0.00	0.00	0.0	3.7	8.23	6.9	16.19			
8	65.4	18.6	80.4	26.9	58.4	14.7	59.0	66.5	18.5	0.00	0.00	0.00	0.0	3.9	8.59	6.2	13.87			
9	66.3	19.1	81.7	27.6	58.7	13.3	59.0	65.2	18.5	0.00	0.00	0.00	0.0	4.3	9.62	6.6	16.36			
10	66.9	19.4	80.4	26.9	51.8	11.0	59.0	66.1	19.0	0.00	0.00	0.00	0.0	3.6	6.73	8.1	18.03			
11	66.4	19.1	80.4	26.9	53.2	11.8	59.0	66.8	19.4	0.00	0.00	0.00	0.0	3.6	7.96	7.3	16.33			
12	66.2	19.0	81.3	27.4	51.4	10.8	59.0	66.4	19.1	0.00	0.00	0.00	0.0	2.7	5.96	6.4	14.79			
13	66.3	19.2	83.3	28.5	54.0	12.2	4.36	66.6	20.4	0.00	0.00	0.00	0.0	2.7	5.97	5.8	12.83			
14	68.4	20.8	79.7	26.5	56.36	56.4	56.4	62.0	22.45	0.05	0.01	0.2	0.01	0.2	2.6	5.91	6.8	15.19		
15	62.2	16.9	75.7	24.3	56.9	13.3	59.0	67.3	17.4	0.00	0.00	0.00	0.0	2.5	5.55	6.0	13.48			
16	66.6	19.4	81.7	27.6	52.7	11.5	59.0	62.7	19.6	0.00	0.00	0.00	0.0	2.4	5.26	6.5	14.63			
17	69.0	20.6	83.7	28.7	55.8	12.2	59.0	69.7	21.0	0.00	0.00	0.00	0.0	2.7	6.11	6.2	13.87			
18	68.6	20.3	83.5	28.6	58.4	13.1	4.40	69.5	20.8	0.00	0.00	0.00	0.0	2.5	5.57	5.7	12.73			
19	65.9	18.8	80.1	26.7	56.5	13.6	59.0	68.3	20.2	0.00	0.00	0.00	0.0	2.7	6.13	5.9	13.29			
20	63.8	17.7	77.0	25.0	56.2	12.9	59.0	66.1	19.0	0.01	1.3	0.04	1.1	0.02	0.4	2.0	4.54	10.0	22.37	
21	65.9	18.8	79.7	26.5	58.8	13.2	3.25	67.7	19.8	0.00	0.00	0.00	0.0	2.8	6.35	5.9	13.29			
22	65.9	18.8	79.7	26.5	52.9	11.6	6.0	66.3	19.2	0.00	0.00	0.00	0.0	2.9	6.49	6.1	13.67			

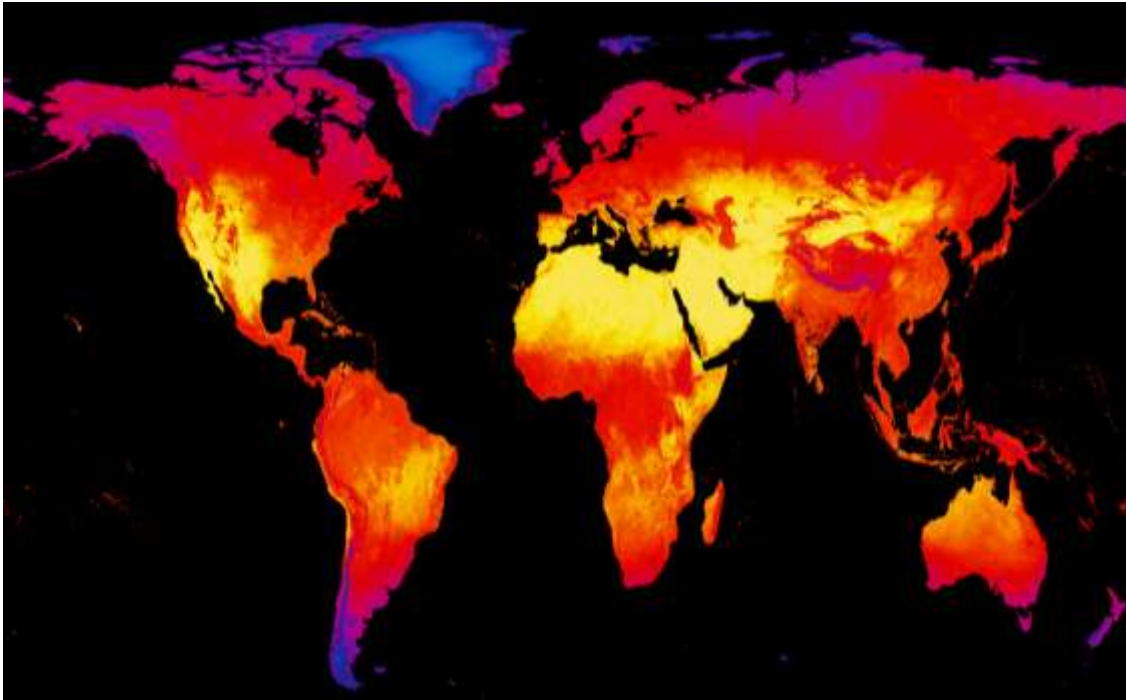


Figure 17: Remote Sensor Land Surface Temperature LST Data



Figure 18: Remote Sensor Sea Surface Temperature SST Data

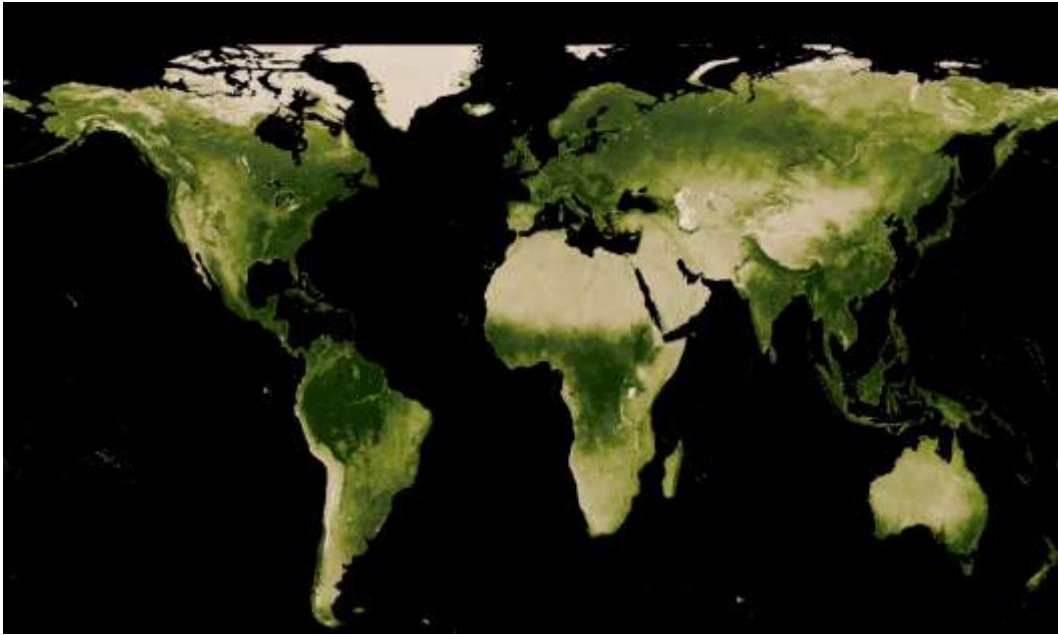


Figure 19: Remote Sensor Normalised Difference Vegetation Index NDVI Data

6.3 Physical Database Model Design

This section presents the physical SQL representation design of the database model which takes into account the logical integration of the entities and constraints that ensure data integrity. Below are few listings of the physical model design of the database, the full prototypical database implementation model design codes are documented in the appendix.

Remote sensor observations (raster coverages) table design and data loading SQL are generated using the raster2pgsql.py. Raster2pgsql.py is a python script which works with GDAL library to generate SQL expressions for creating raster tables and loading the raster data. Raster2pgsql.py takes input parameters such as;

-r RASTER, which is the raster data to be loaded, at least one raster file, is mandatorily required.

-t TABLE, this is the raster destination table, mandatory parameter too.

-s SRID, -b BAND, -k BLOCK_SIZE and so on are optional parameters needed but very important parameters.

The raster table physical design models were generated with the raster2pgsql.py script and afterwards manually customized to ensure that the tables comply with the logical database model design.

For example in creating and loading up the NDVI raster table from MODIS sensor, the raster2pgsql.py script was used in the command line as thus:

```
Raster2pgsql.py -r D:/Data/ndvi.tif -t ndvi -s 4326 -l -k 30x30 -o D:/Data/ndvi.sql
```

Where:

- -r is the input raster (ndvi) located at D:/Data/ndvi.tif
- -t is the table name to be created (ndvi)
- -s is the SRID (EPSG code) of the raster, in this case it is 4326 WGS84 lat-long
- -l is for spatial indexing of the data
- -k is used to specify the size of each tile for breaking the input raster into tiles
- -o is the output raster sql file (D:/Data/ndvi.sql), this contains the sql file for creating and loading up the table in PgAdmin.

For this example, appendix C shows how the output sql file looks like, a multi-dimensional array of discrete data in WKB (World Known Binary) format.

After the generating the tables and data with the Raster2pgsql.py script, the tables were manually customized to comply with the necessary facilities and constraints in the logical design. For the SQL codes of the physical implementation of some of these remote sensor observation tables and logical design see Appendix A.

For the in-situ sensor observation and other tables in the model, the SQL codes for creation of the tables and insertion of values carried out in this prototypical implementation exercise are also documented in the appendixes A and B.

Figure 20 below is a screen shot excerpt showing the physical implementation of the database model in PostgreSQL/PostGIS2.0 database management system. Both the remote and in-situ sensor observations efficiently stored for seamless integration.

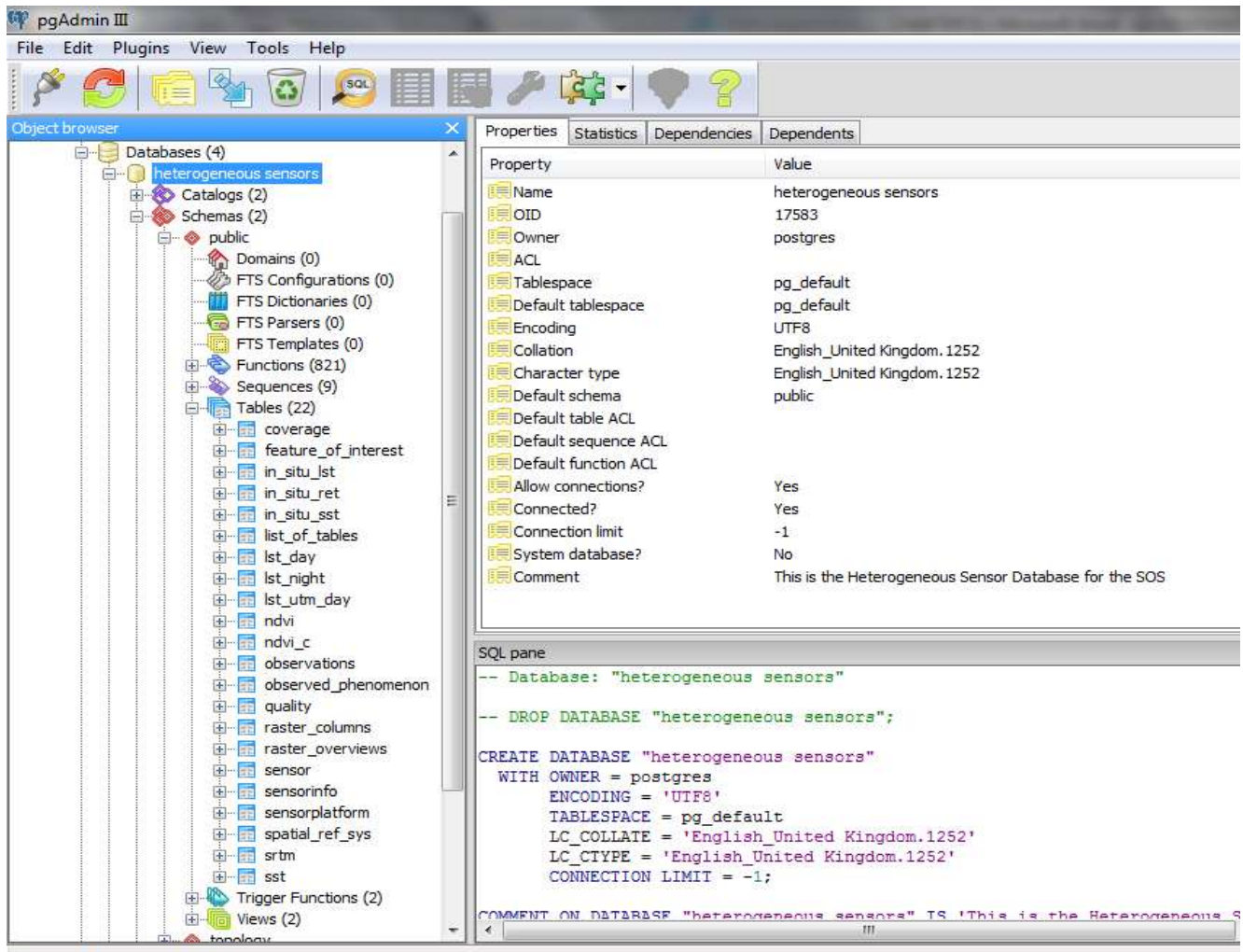


Figure 20: A screen shot excerpt of the heterogeneous sensor database model with the tables

6.3 Implementation of some of the Application Scenarios and Use Cases.

In this section we would be implementing and testing some of the scenarios and use cases we discussed in chapter 5. We would be running the queries from within the OpenJump desktop application. We connect to the database from the OpenJump desktop application and run the queries from there.

6.3.1 Scenario 1: In-situ and satellite surface temperature analysis

This scenario calculates the temperature difference between the in-situ sensor land surface temperature observation and remote sensor land surface temperature observation of a

particular location. Listing 8 is used to obtain the required result from within the OpenJump desktop application.

Listing 8 : Scenario 1 implementation SQL code

```
SELECT val1, (gv).val AS val2 ,val1-(gv).val AS  
diffval,geom  
  
FROM ( SELECT ST_intersection(rast,the_geom) AS gv,  
temp_value AS val1, ST_AsBinary(the_geom) AS geom  
FROM in_situ_lst , lst_day  
WHERE the_geom && rast  
AND ST_intersects(rast,the_geom)  
AND temp_lst_id = 1  
) foo;
```

Here, this query picks up a particular temperature observation from the in-situ land surface temperature 'val1', in-situ_lst table of a location where id = 1, compares the temperature value with the corresponding remotely observed temperature, 'val2' of that same location on the raster temperature coverage, Lst_day and returns the difference, 'diffval'.

Figure 21 below is the implementation screen short excerpts from the OpenJump desktop application.

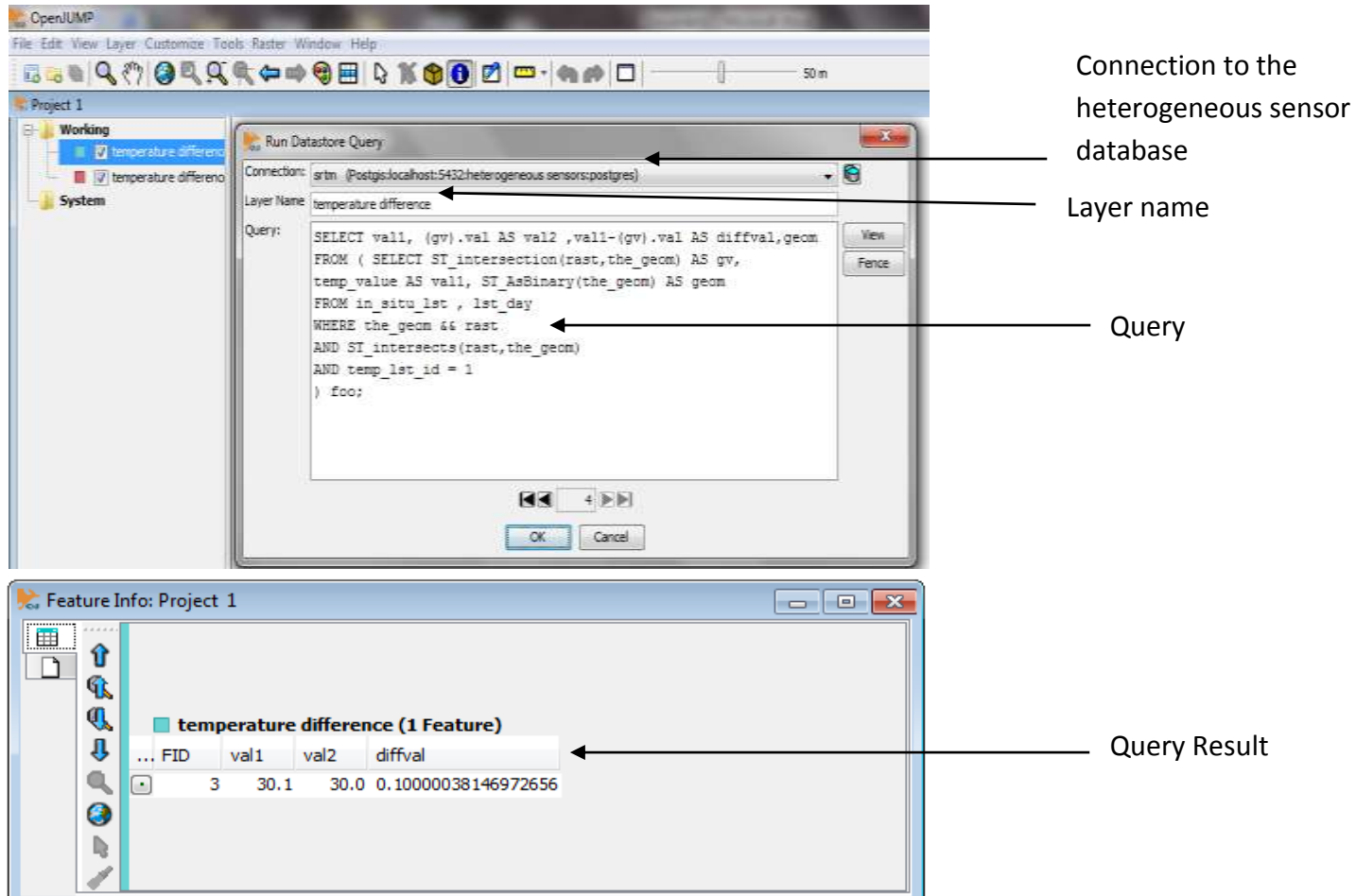


Figure 21: Screen short excerpt of Scenario 1 implementation in OpenJump

6.3.2 Scenario 2.1: Weighted Mean surface temperature values from a vector buffer

In chapter 5, scenario 2.1, we described a scenario where, we could select a particular observation in the in-situ temperature observation, create a buffer of a given radius around that observation, then overlap this buffer geometry on the raster temperature coverage and obtains a weighted mean surface temperature value within the buffered region from the raster coverage.

Listing 9 below is used to obtain the required result from within the OpenJump desktop application.

Listing 9: Scenario 2.1 implementation SQL code

```
SELECT sum(ST_Area(the_geom)*val)/(sum(ST_Area(the_geom)))
AS meantemp

FROM (SELECT(ST_Intersection(R.rast,(ST_Buffer(I.the_geom,
500)))).geom AS the_geom,

(ST_Intersection(R.rast,(ST_Buffer(I.the_geom, 500)))).val AS val

FROM in_situ_lst I, lst_day R

WHERE the_geom && R.rast

AND ST_Intersects(R.rast,the_geom)

AND I.temp_lst_id = 2

) foo;
```

In this case the query select the in-situ temperature observation, from the in_situ_lst table where id=2, runs a buffer of 500m radius , then overlaps this buffer geometry on the raster temperature coverage, lst_day and calculates the weighted mean temperature value within this buffer region from the raster temperature coverage.

Figure 22 is the result of this particular query from our database.

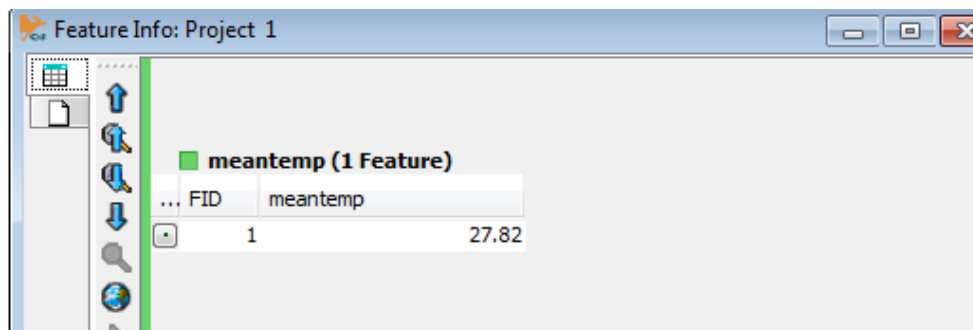


Figure 22: Screen short excerpt of a sample Scenario 2.1 implementation result in OpenJump

6.3.3 Scenario 2.2: Day and Night time temperature Difference of a Location

This is a case as we described in chapter 5, where one could be interested in the day and night time temperature difference from the satellite observation of a particular location contained in an in-situ sensor observation table.

We implemented this scenario from within our prototypical that database using this sample SQL code in listing 10 below.

Listing 10: Scenario 2.2 implementation SQL code

```
SELECT
    DT, NT, DT-NT as TD, the_geom
FROM    (SELECT ST_Value(R1.rast,I.the_geom) AS DT,
            ST_Value(R2.rast,I.the_geom) AS NT,
            ST_AsBinary(I.the_geom) as the_geom
        FROM in_situ_lst I, lst_day R1, lst_night R2
        Where I.temp_lst_id = 2
        AND ST_Value(R1.rast,I.the_geom) IS NOT NULL
        AND ST_Value(R2.rast,I.the_geom) IS NOT NULL
        ) foo;
```

Here the query identifies the in-situ observation from in_situ_lst temperature coverage where id =2, intersects the geometry of that observation on the day, lst_day and night, lst_night raster temperature coverages and returns the respective temperatures values and their difference from raster coverages.

FID	dt	nt	td
3	35.354331970214844	10.551180839538574	24.80315113067627

Figure 23: Screen short excerpt of a sample Scenario 2.2 implementation result in OpenJump

Figure 23 shows the query result from within the OpenJump application.

6.3.4 Scenario 3: Geo-scientific Analysis of In-situ Temperature /Air Pressure Data and Satellite Elevation/ Height Observation

This is a scenario as described in chapter 5, where a geo-scientist wants to ascertain the surface elevation or height factor of a particular temperature or air pressure observation. This could be to verify the correlation between height and temperature.

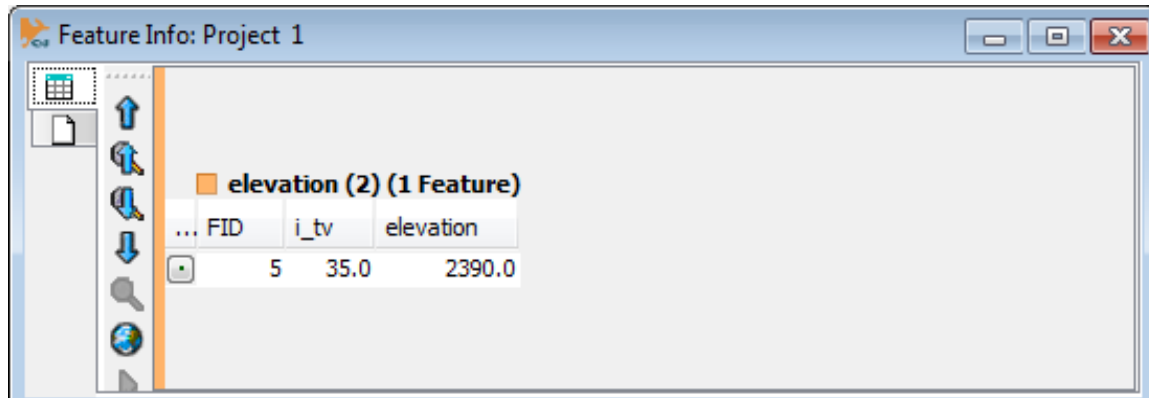
In the query below, we used the geometry of the temperature observation, `in_situ_lst`, where `id = 2` to intersect on the SRTM raster elevation coverage and obtained the height of that point.

Hence this was implemented using these few lines of SQL codes in listing 11 below.

Listing 11: Scenario 3 implementation SQL code

```
SELECT I_tv, elevation, the_geom
FROM (SELECT I.temp_value as I_tv, ST_Value(R.rast,
I.the_geom) as
    elevation, ST_AsBinary(I.the_geom) as the_geom
FROM in_situ_lst I, srtm R
WHERE ST_Value(R.rast, I.the_geom) IS NOT NULL
AND I.temp_lst_id = 2
```

Figure 24 below is the query result from within the OpenJump application.



The screenshot shows a window titled 'Feature Info: Project 1'. On the left is a vertical toolbar with icons for map navigation and information. The main area displays a table of data for a feature named 'elevation (2) (1 Feature)'. The table has three columns: 'FID', 'i_tv', and 'elevation'. A single row of data is visible with values 5, 35.0, and 2390.0.

...	FID	i_tv	elevation
	5	35.0	2390.0

Figure 24: Screen short excerpt of a sample Scenario 3 implementation result in OpenJump

6.3.5 Scenario4: Estimation of Actual Crop Evapotranspiration ET

This scenario as described in chapter 5, where we have in-situ Reference Evapotranspiration RET observations from weather automatic stations and NDVI raster coverage of the same area. Hence Actual Evapotranspiration could be estimated from an aggregation of RET and Fraction of Vegetation cover FVC, where FVC is derived from NDVI.

In the query below, a geo-scientist wants to obtain the Actual Evapotranspiration AET of a particular location, having the Reference Evapotranspiration RET of that particular point on the in-situ observation table and the NDVI coverage of the area as well contained in the same database as we are proposing.

To implement this scenario as we described in chapter 4, we could use 1 and 0 as the approximate maximum and minimum NDVI values respectively within the area, this would give us an approximate estimation not very précised.

But to obtain the actual NDVImax and NDVImin of the coverage area, we used the SQL query below in listings 12 and 13, which were later used in the estimation of the AET.

Listing 12: SQL Query to obtain the NDVI_{max}

```
SELECT (stats).max
FROM (SELECT ST_SummaryStats(rast) As stats
      FROM ndvi
      ORDER by stats DESC
      limit 1 ) As foo;
```

Listing 13: SQL Query to obtain the NDVI_{min}

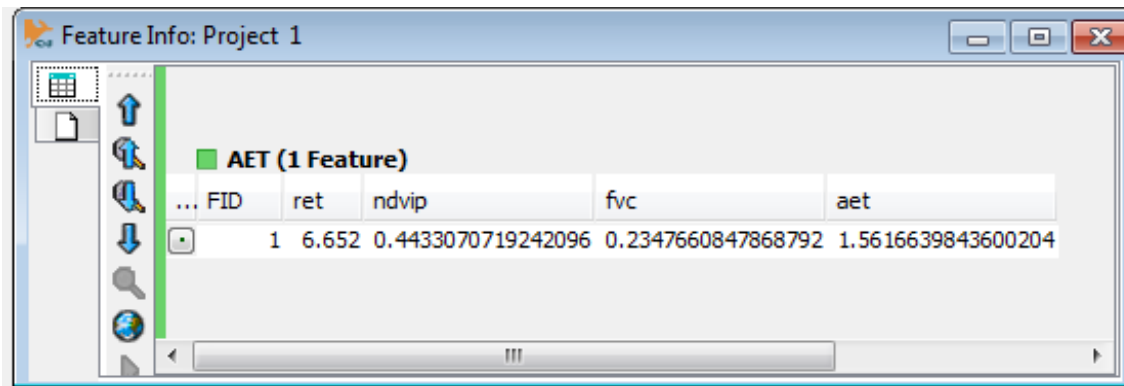
```
SELECT (stats).min
FROM (SELECT ST_SummaryStats(rast) As stats
      FROM ndvi
      ORDER by stats ASC
      limit 1 ) As foo;
```

Afterwards we estimated the AET of a point in the RET, in_situ_ret table where id =1 by implementing the SQL code in listing 14. From the query in listing 21 and 22, we obtained the NDVI_{max} and NDVI_{min} as 0.86 and 0 respectively and input them in the query below.

Listing 14: Scenario 4 implementation SQL code

```
SELECT RET, NDVIp,(pow(((NDVIp-0.86)/(0.86-0)),2)) as FVC,
(pow(((NDVIp-0.86)/(0.86-0)),2))*RET as AET, the_geom
FROM (SELECT ST_Value(R.rast,I.the_geom) as NDVIp, I.value as RET,
ST_AsBinary(I.the_geom) as the_geom
FROM in_situ_ret I, ndvi R
WHERE ret_id = 1
AND ST_Value(R.rast,I.the_geom) IS NOT NULL) foo;
```

Here in Figure 25 is the result of the query from OpenJump client end.



FID	ret	ndvip	fvc	aet
1	6.652	0.4433070719242096	0.2347660847868792	1.5616639843600204

Figure 25: Screen short excerpt of a sample Scenario 4 implementation result in OpenJump client end

6.3.6 Scenario 5: DescribeCoverage (Raster Coverage Metadata) Query Operation

This scenario as we described in chapter 5 is a common practice in the OGC SWE, used by the client to describe any raster coverage contained in the service database. In this case, we are leveraging the PostGIS raster metadata description function to provide the client side description of a raster coverage through an SQL query.

Using the SQL code in listing 15 below, we could describe some basic metadata of the SRTM coverage in our database.

Listing 15: Scenario 5 implementation SQL code

```
SELECT (md).*, (bmd).*, the_geom
FROM (SELECT ST_Metadata(R.rast) AS md,
      ST_BandMetadata(R.rast) AS bmd,
      ST_AsBinary(R.rast::geometry) as the_geom
FROM srtm R
LIMIT 1
) foo;
```

Figure 26 below shows the result of this query obtained from within the client application, the OpenJUMP.

Describe SRTM (1 Feature)


...	FID	upperleftx	upperlefty	width	height	scalex	scaley	skewx	skewy	srtd	numbands	pixeltype	hasnodatavalue	nodatavalue	iso
	2	-105.25195454545454	38.83113636363637	50	50	2.727272727272727E-4	-2.727272727272727E-4	0.0	0.0	4326	1	16BSI	f	1.0075959E-37	f

Figure 26: Screen short excerpt of a sample Scenario 5 implementation result in OpenJump client end

In conclusion of this chapter, the implemented scenarios have so far shown that we could basically do geo-processing and analysis between in-situ and remote sensor observations at the database backend. In other words, vast kinds of geo-processing on the geospatial web services could now be done completely at the Sensor Observation Service database backend, therefore reducing extensively, massive workload on the service middleware of the service.

6.4 Performance and Hypothesis Evaluation

The aim of this chapter is to describe and discuss the practical impact and evaluation of this proposed heterogeneous sensor database model in the Sensor Observation Service in terms of performance and the results of the real life scenarios we carried out in chapter 5. Hence we discuss and evaluate the hypothesis of this thesis in relation to our practical implementation performance, how the model goes to reduce work and communication load on the service middleware and how flexible it is for client side operations.

6.3.1 Performance Analysis

The performance analysis of this proposed heterogeneous sensor database model is based on heterogeneous data integration capability, speed, size and flexibility and cost.

Integration Capability

The conceptual, logical and physical design and model of this database based on PostgreSQL/PostGIS 2.0 functionalities is excellent in term of seamless raster and vector data integration and processing. The concept of storing the in-situ and remote sensor observations as vector and raster coverages , leveraging the corresponding geometry columns for integration worked very well.

Storage and Analysis Speed

This prototypical implementation exercise was performed on a TOSHIBA machine with 2 GHz INTEL processor and 4GB of RAM running on window 7 platform. It is good to keep the machine capability in mind when evaluating the storage and analysis execution speed of this database model implementation.

In the implementation exercise in this chapter, the SRTM elevation raster coverage was the highest in size among all the data we used in the exercise. The size of the SRTM SQL file generated with the GDAL-Python raster loader was 169MB.

This 169MB SQL file of 9213 rows of multi-dimensional arrays of discrete data was loaded into the database in 73088 ms (1.22 minutes).

Two of the scenarios we implemented in chapter 5 that involved more geo-processing and analysis on the vector and raster coverages are scenario 2.1, weighted mean surface temperature within a buffer region and scenario 4, estimation of Actual Evapotranspiration.

Scenario 2.1 was executed in 1438ms (1.438 seconds) and scenario 4, executed in 341ms (0.341 seconds).

Looking at the storage speed and analysis execution speed on massive raster and vector datasets is very impressive and commendable. The use of Gist spatial indexing is the key for the impressive execution speed. It is also important to keep in mind that, the PostGIS2.0 database extension we used for the prototypical exercise is a beta (trial) version which was released in August 2011, still in development stage. The full version which is expected to be out in the first quarter of 2012 will have more powerful capabilities in terms of data loading and analysis.

Hopefully it will come as a full standalone extension with fewer configurations unlike what we had to configure in this implementation.

Query Flexibility

The various geo-processing scenarios we implemented in this chapter from within the OpenJump client side desktop application show that, this approach of delivering SQL based queries from the client end direct to the database backend makes it more flexible for the user on the client end to deliver geo-processing queries of extensive complexity involving in-situ and remote sensor observations. This extensive support for different kinds of geo-processing and analysis involving in-situ and remote sensor observations through native SQL queries makes this approach advantageous to the current approach of having different geo-processing modules on the web service for specific purposes. In that case users are restricted only to the specific capability the service offers.

Storage Size and Cost

The proposed heterogeneous database model is implemented in PostgreSQL/PostGIS database management system which is an Open Source RDBMS. It is free and easy to install and deploy. The maximum database storage size is unlimited, maximum table size is 32 Terabytes, Row Size, 1.6 Terabytes, field size, 1GB, Rows per table is unlimited, Columns per table is 250-1600 depending on the column types and maximum Indexes per table is unlimited (32). These properties in terms of size and cost give an advantage and leave us with unlimited sensor observation storage and management. That is to say, unlimited storage and management provision is in place for large and massive remote sensor observations.

6.3.2 Research Hypothesis Evaluation

In this section, we would be evaluating our research hypothesis which is “If remote and in-situ sensor observations can be effectively integrated at the database backend, then communication and work load on the service middleware of the Geo-Web Services can be drastically reduced and query requests from the users would be more flexible” based on results and achievement of this research so far.

In view of this, we would be evaluating how the research results so far has supported the following claims contained in the hypothesis:

- reduced communication load on the service
- reduced work load and massive data retrieval on the service middleware
- and more flexible query requests on the user end.

Reduced communication load

Currently as we saw in (11), (1), (13), (14) and as described in figure 27 below, geo-processing and spatial analytics involving in-situ and remote sensor observations in dynamic systems (such as in geo-web services) entail back and forth communication to different ends of databases. In-situ and remote sensor observations were usually stored separately in different databases. Remote sensor observations (raster) mainly stored as flat files and mostly accessed through the File Transfer Protocol FTP.

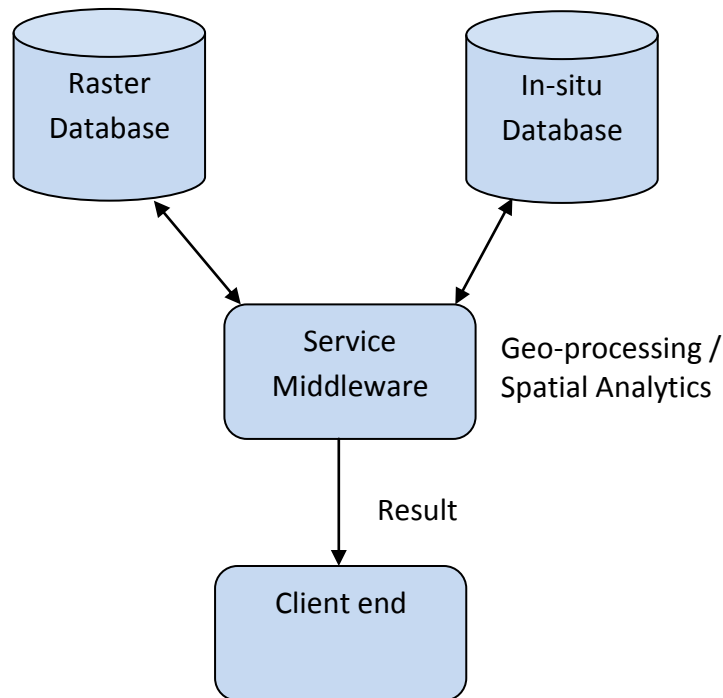


Figure 27: Current workflow in a dynamic geo-processing service

In the contrary, so far in this thesis, the prototypical implementation our proposed heterogeneous sensor database model and the model of how it can be integrated seamlessly with the OGC geo-web services show that these disparate sensor observations can be integrated and managed in a single spatial database leveraging PostGIS functionalities. Hence communication load to different databases reduced invariably as shown in figure 28 below. The

communication time lag incurred in the downloading of raster images from a flat file database via ftp, obtaining in-situ observations from an in-situ sensor observation service and integrating the two on the service middleware level is invariably reduced greatly adopting this heterogeneous database approach. We will provide the quantitative support to this claim as a further work. The time frame of this thesis didn't permit us to lay a hand on a running web service that runs on the multiple database approach such as in (11), (1), (13), (14) etc. to obtain a quantitative time lag comparison to our direct approach as shown in fig 28 below.

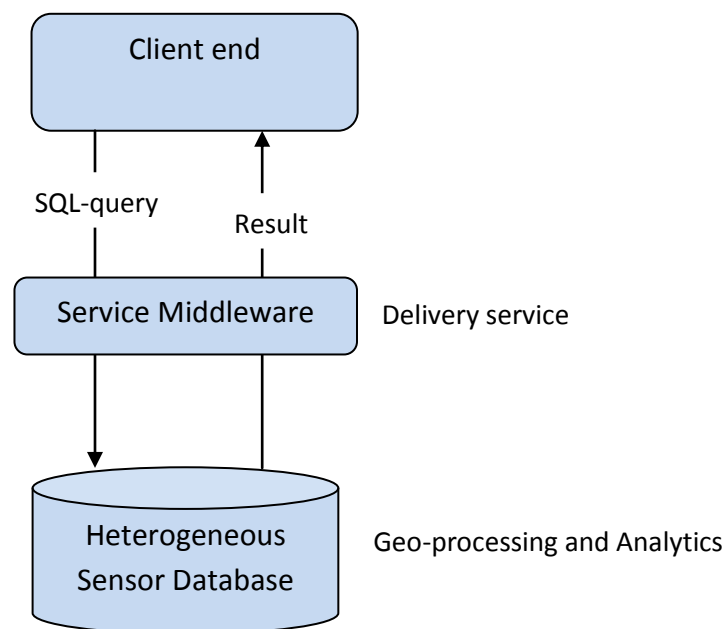


Figure 28: Proposed workflow in a dynamic geo-processing service by means of the heterogeneous sensor database

Reduced work and massive data retrieval load

Also taking a look at the contents and the processes in dynamic systems such as in (11), (1), (13), (14) and in the OGC Web Processing Services, they provide clients access and results based on pre-programmed calculations and/or computation models that operate on the spatial data. To enable geospatial processing and operations of diverse kinds, from simple subtraction and addition of sensor observations (e.g. the difference between satellite observed temperature

and in-situ observation of a location) to complicated ones such as climate change models, requires the development of a wide variety of models on the service middleware. This is massive in work load and huge amount of programming on the service. Also the data required for these services are usually retrieved dynamically from different databases and services as described in figure 27. This most times entails massive data retrieval especially from the satellite data (raster) storage.

Contrarily, by the means of a heterogeneous sensor database model as developed and implemented in this thesis leveraging the functionalities of PostGIS 2.0 database extension, geo-processing and analytics involving remote and in-situ sensor data are carried out at the database backend by native SQL request statements. Therefore the variety of geo-processing work load on the service middleware is reduced. As described in figure 28 above, the service middleware in our case is majorly for service delivery from the client to database and vice versa. Massive data retrieval before processing is completely avoided. Also massive programming involved in the development of different kinds of geo-processing models on the web service is reduced.

Flexibility in query request from the user end

Language based query request such as the (SQL) has been considered advantageous especially by the database community because is very flexible, declarative, optimizable and more safe in evaluation (9). The current approach of deploying specific geo-processing models for specific tasks as a service on the geo-web service do not offer the flexibility obtainable from SQL-enabled query request system like the Web Query Service WQS proposed in this thesis. The SQL query request approach gives the users the leverage to construct queries of disparate simplicity or complexity to get the required results.

Therefore we have so far in this research arguably provided that the hypothesis of this research is valid. It is true and valid to say that effective and efficient integration of remote and in-situ sensor observations at the database backend such as modeled and implemented in this thesis would reduce communication and work load on the service middleware of the geo-web services and as well make query request from the user end more flexible.

Chapter 7

7.0 CONCLUSION AND FUTURE WORK

In summary, the main essence of this research is to prove that disparate sensor data from heterogeneous sensors can be integrated in a single database schema in the sensor observation service which gives the leverage to reduced communication and massive work load at the service middleware and makes query request a lot more flexible from the user end by means of SQL query request statements.

In view of that, we started in this thesis by presenting an introduction to the research concept, which is anchored on the development of a heterogeneous sensor database model for the Sensor Observation Service SOS whereby geo-processing and analytics could now be done on the database backend and not necessarily on the service middleware.

We carried out a thorough background research on studies that dealt with in-situ sensor observation service, remote sensor observation (raster) databases, integration of remote sensor and in-situ sensor observations for environmental monitoring and management. Also we did a good background search on the OGC SWE, SOS, WCS and WCPS to have a good understanding of how they work. All these background research are concisely documented in chapter 2.

Then we carried out a detailed requirement analysis on the best approach to model conceptually and logically a heterogeneous sensor database model that allows for the integration and analysis of disparate sensor observations (in-situ and remote) at the database end. We discovered the power of PostGIS 2.0 extension of PostgreSQL database management system in implementing this type sensor database model.

The outcome of the requirement analysis helped us to conceptually and logically model a heterogeneous sensor database schema. The conceptual model was done in Unified Modeling Language UML and abstracted the Entity Relationship logical model from the UML conceptual model. We developed a detailed conceptual model on how the proposed heterogeneous sensor database model can be integrated seamlessly with other geo-web services in the SOS. We

hereby discussed the concept of Web Query Service WQS for query delivery service from the client to the database backend.

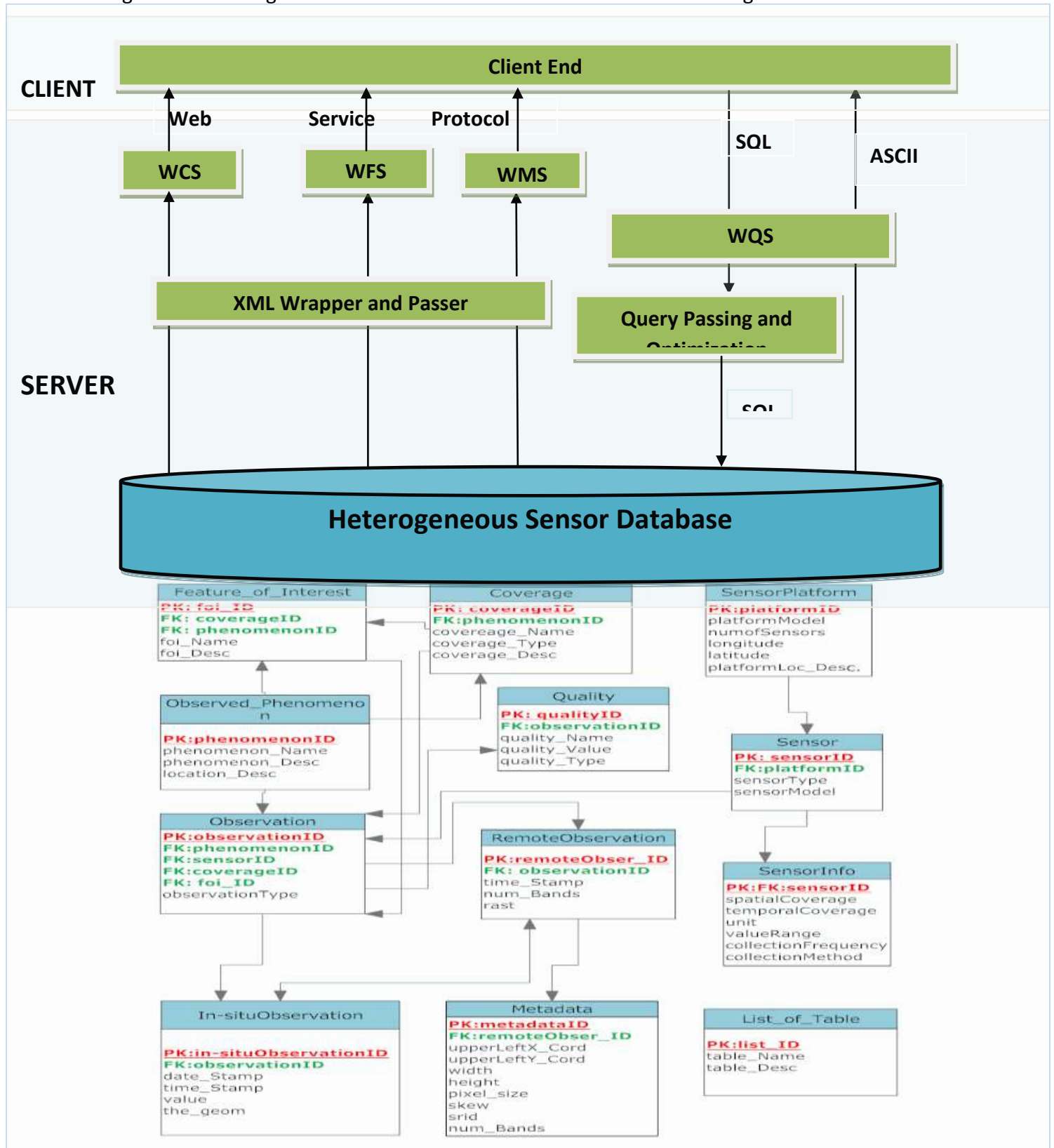
We discussed and described scenarios/ use cases where this type of sensor database can be leveraged to accomplish geo-scientific analysis and processing involving in-situ and remote sensor observations. Example queries were also formulated and analyzed.

Then we prototypically implemented this heterogeneous sensor database model in PostgreSQL/PostGIS 2.0 database management system. The scenarios were implemented and tested from within a client application, the OpenJump application. The performances of the scenarios in the database context were very good.

The practical usefulness of this proposed approach will be very more appreciated when the model of integrating the proposed sensor database and other geo-web services in the SOS is fully implemented. Because of the limited time frame for this master's thesis, we were not able to implement the integration model in the SOS.

As a future work, the full integration of this heterogeneous sensor database model with other geo-web service will be implemented as shown in figure 29 below. We discussed the implementation details in chapter 4. Therefore it remains a further implementation exercise.

Figure 29: Heterogeneous sensor database and Geo-web services integration model



References

1. *Development of a Dynamic Web Mapping Service for Vegetation Productivity Using Earth Observation and in situ Sensors in a Sensor Web Based Approach*. **Kooistra, L., et al.** 4, 2009, *Sensors*, Vol. 9, pp. 2371-2388.
2. **Hamre, Torill**. Integrating Remote Sensing, In Situ and Model Data in a Marine Information System (MIS). *Marine Information System (MIS)*, in *Proc. Neste Generasjons GIS*. 1993, pp. 181-192.
3. *A flexible geospatial sensor observation service for diverse sensor data based on Web service*. **Chen, Nengcheng, et al.** 2, March 2009, *ISPRS Journal of Photogrammetry and Remote Sensing*, Vol. 64, pp. 234-242. DOI: 10.1016/j.isprsjprs.2008.12.001. ISSN 0924-2716.
4. **Baumann, Peter and Chulkov, Georgi**. *Web Coverage Processing Service (WCPS)*. Bremen : School of Engineering and Science, Jacobs University Bremen, 2007. Technical Report.
5. *LARGE-SCALE MULTIDIMENSIONAL COVERAGE DATABASES*. **Baumann, Peter, et al.** San Antonio : s.n., March 2-5, 2003. 26th GITA Annual Conference.
6. *Management of multidimensional discrete data*. **Baumann, Peter**. Issue 4, October 1994, *The VLDB Journal*, Vol. Volume 3, pp. 401-44.
7. *Efficiently managing large-scale raster species distribution data in PostgreSQL*. **Zhang, Jianting, Gertz, Micheal and Gruenwald, Le**. New York, NY, USA : ACM, 2009. GIS '09 Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ISBN: 978-1-60558-649-6.
8. **Baumann, Peter**. Large-Scale Earth Science Services: A Case for Databases. *Advanced in Conceptual Modelling- Theory and Practice*. s.l. : Springer Berlin/Heidelberg, 2006, pp. 75-84.
9. *Designing a Geo-scientific Request Language - A Database Approach*. **Baumann, Peter**. s.l. : Springer-Verlag Berlin, Heidelberg, 2009. SSDBM 2009 Proceedings of the 21st International Conference on Scientific and Statistical Database Management. ISBN: 978-3-642-02278-4.
10. *Store, manipulate and analyze raster data within the PostgreSQL/PostGIS spatial database*. **Racine, Pierre**. Denver : <http://2011.foss4g.org>, 2011. FOSS4G.
11. **Rueda, Carlos and Gertz, Michael**. Real-Time Integration of Geospatial Raster and Point Data Streams. *Statistical and Scientific Database Management*. 2008, pp. 605--611.
12. **NERSC**. WARMER Web-GIS Portal. [Online] [Cited: August 26, 2011.] <http://warmer.nersc.no/index.xml>.
13. *WARMER in-situ and remote data integration*. **Alastair Allen, et al.** Southampton (UK) : s.n., 30th March 2009. National Oceanography Center.

14. *An integrated Earth sensing sensorweb for improved crop and rangeland yield predictions.* **Teillet, P M, et al.** 2007, Canadian Journal of Remote Sensing, Vol. 33, pp. 88-98.
15. *Data and monitoring needs for a more ecological agriculture.* **J, Kucharik Christopher and M, David Zacks P.** 2011, IOPScience, p. Environ. Res. Lett. 6.
16. *The SEQUOIA 2000 storage benchmark.* **Stonebreaker, Micheal, et al.** New York, NY, USA : ACM, 1993. SIGMOD '93 Proceedings of the 1993 ACM SIGMOD international conference on Management of data. pp. 592-595.
17. **Botts, M., et al.** *Overview and high level architecture.* s.l. : OGC sensor web enablement, 2007. Technical report, OGC.
18. **OGC, Open Geospatial Consortium.** *OGC® WCS 2.0 Interface Standard - Core.* s.l. : OGC, 2010. Technical report, OGC 09-110r3.
19. **OGC, Open Geospatial Consortium.** *OGC® GML Application Schema - Coverages.* s.l. : OGC, 2010. Technical Report OGC 09-146r1, Version: 1.0.0.
20. *Geographic information -- Schema for coverage geometry and functions.* **ISO.** 2009, TC 211 - Geographic information/Geomatics.
21. **PostGISWiki.** PostGIS UsersWiki. [Online] [Cited: September 10, 2011.] <http://trac.osgeo.org/postgis/wiki/UsersWikiCoveragesAndPostgis>.
22. **Stasch, Christoph.** SosDataModeling. *52north.orgWiki* . [Online] [Cited: September 14, 2011.] <https://wiki.52north.org/bin/view/SensorNet/SosDataModeling>.
23. **NOAA.gov.** Shipboard Sensor Database. *Shipboard Sensor Database (SSD) Program.* [Online] [Cited: September 14, 2011.] <http://www.nodc.noaa.gov/cgi-bin/ssd/ssd.pl>.
24. **postgis.refractions.net.** PostGIS 1.5.3 Manual. *postgis.refractions.net Web site.* [Online] [Cited: August 20, 2011.] <http://www.postgis.org/docs/>.
25. *PostGIS WKT Raster. An Open Source alternative to Oracle GeoRaster.* **Arevalo, Jorge.** Barcelona : s.n., 2010. FOSS4G .
26. *Groundwater and Vegetation Effects on Actual Evapotranspiration Along the Riparian Zone and of a Wetland in the Republican River Basin.* **Gregory Cutrell, M. Evren Soylu.** Nebraska-Lincoln : s.n., 2009.
27. **Obe O. Regina, Leo S. Hsu.** *PostGIS in Action.* Stamford : Manning Publications Co., 2011. p. 372. ISBN: 9781935182269.
28. **Pierre, Racine.** WKTRasterTutorial01. *PostGIS.* [Online] June 2010. [Cited: 11 12, 2011.] <http://trac.osgeo.org/postgis/wiki/WKTRasterTutorial01>.

29. GDAL . *GDAL - Geospatial Data Abstraction Library*. [Online] July 2011. [Cited: 11 15, 2011.] <http://www.gdal.org/>.
30. NumPy. [Online] [Cited: 11 15, 2011.] <http://numpy.scipy.org/>.
31. **jorgearevalo**. How to install and configure PostGIS Raster on Windows. *GIS4Freedom Blog*. [Online] 3 2011. [Cited: 11 15, 2011.] <http://libregis.org/2011/03/10/how-to-install-and-configure-postgis-raster-on-windows/#comment-433>.
32. **Postgresql.org**. PostgreSQL. *www.postgresql.org*. [Online] [Cited: December 13, 2011.] <http://www.postgresql.org/about/>.

Appendixes

Appendix A

SQL implementation codes for physical design and creation of the prototypical heterogeneous sensor database and the tables in PostgreSQL/PostGIS2.0.

```
CREATE DATABASE "heterogeneous sensors"
WITH OWNER = postgres
ENCODING = 'UTF8'
TABLESPACE = pg_default
LC_COLLATE = 'English_United Kingdom.1252'
LC_CTYPE = 'English_United Kingdom.1252'
CONNECTION LIMIT = -1;
```

```
CREATE TABLE lst_day (  
    rid serial NOT NULL,  
    filename text,  
    rast raster,  
    observed_date date,  
    CONSTRAINT lst_day_pkey PRIMARY KEY (rid),  
    CONSTRAINT enforce_srid_rast CHECK (st_srid(rast) = 4326))  
WITH ( OIDS=FALSE  
);  
ALTER TABLE lst_day OWNER TO postgres;  
Index: lst_day_rast_gist_idx
```

```
CREATE TABLE list_of_tables (  
    list_id INTEGER NOT NULL,  
    table_name VARCHAR (100),  
    table_desc VARCHAR (100),  
    CONSTRAINT list_of_tables_pkey PRIMARY KEY  
    (list_id)  
    );
```

```

CREATE TABLE ndvi(
rid serial NOT NULL,
filename text,
rast raster,
observed_date date,
obs_id integer,
CONSTRAINT ndvi_pkey PRIMARY KEY (rid),
CONSTRAINT fk_ndvi FOREIGN KEY (obs_id)
REFERENCES observations (obs_id) MATCH FULL
ON UPDATE NO ACTION ON DELETE NO ACTION,
CONSTRAINT enforce_srid_rast CHECK (st_srid(rast) = 4326));
ALTER TABLE ndvi OWNER TO postgres;

Index: ndvi_rast_gist_idx
-- DROP INDEX ndvi_rast_gist_idx;

CREATE INDEX ndvi_rast_gist_idx ON ndvi USING gist
(st_convexhull(rast));

```

```

CREATE TABLE observed_phenomenon
(
pheno_id integer NOT NULL,
pheno_name character varying(50),
pheno_desc character varying(50),
location_desc character varying(100),
CONSTRAINT pheno_pkey PRIMARY KEY (pheno_id)
);

```

```

CREATE TABLE in_situ_ret (
    ret_id integer NOT NULL,
    obs_id integer NOT NULL,
    time_stamp date,
    "value" real,
    the_geom geometry(Point,4326),
    CONSTRAINT in_situ_ret_pkey PRIMARY KEY (ret_id),
    CONSTRAINT fk_in_situ_ret FOREIGN KEY (obs_id)
        REFERENCES observations (obs_id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE NO ACTION)
WITH ( OIDS=FALSE);
ALTER TABLE lst_day OWNER TO postgres;
Index: lst_day_rast_gist_idx

```

```

CREATE TABLE coverage (
    covid serial NOT NULL,
    coverageName VARCHAR (45),
    coverageType VARCHAR (20),
    pheno_id INTEGER,
    CONSTRAINT coverage_pkey PRIMARY KEY (covid),
    CONSTRAINT enforce_coverageType CHECK (coverageType IN
('raster', 'vector')),
    CONSTRAINT fk_cov FOREIGN KEY (pheno_id) REFERENCES
Observed_Phenomenon(pheno_id)
);

```

```

CREATE TABLE in_situ_sst (
    temp_id integer NOT NULL,
    obs_id integer NOT NULL,
    time_stamp date,
    "value" real,
    the_geom geometry(Point,4326),
    CONSTRAINT in_situ_sst_pkey PRIMARY KEY (temp_id),
    CONSTRAINT fk_in_situ_sst FOREIGN KEY (obs_id)
        REFERENCES observations (obs_id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE NO ACTION)
WITH ( OIDS=FALSE);
ALTER TABLE lst_day OWNER TO postgres;
Index: lst_day_rast_gist_idx

```

```

CREATE TABLE feature_of_interest
(
    foi_id serial NOT NULL,
    covid INTEGER NOT NULL,
    foi_Name VARCHAR (45),
    foi_Desc VARCHAR (50),
    pheno_id INTEGER,
    CONSTRAINT foi_pkey PRIMARY KEY (foi_id),
    CONSTRAINT fk1_foi FOREIGN KEY (covid) REFERENCES
coverage(covid),
    CONSTRAINT fk2_foi FOREIGN KEY (pheno_id)
REFERENCES Observed_Phenomenon(pheno_id)
);

```

```
CREATE TABLE sensorPlatform(  
  platform_id INTEGER NOT NULL,  
  platform_Model VARCHAR (50),  
  num_sensors INTEGER NOT NULL,  
  longitude REAL,  
  latitude REAL,  
  platform_loc_desc VARCHAR (100),  
  CONSTRAINT sensorPlatform_pkey PRIMARY KEY (platform_id),  
);
```

```
CREATE TABLE sensor(  
  sensor_id INTEGER NOT NULL,  
  platform_id INTEGER NOT NULL,  
  sensor_type VARCHAR (20),  
  sensor_model VARCHAR (50),  
  CONSTRAINT sensor_pkey PRIMARY KEY (sensor_id),  
  CONSTRAINT fk_sensor FOREIGN KEY (platform_id)  
  REFERENCES sensorplatform(platform_id),  
  CONSTRAINT enforce_sensor_type CHECK (sensor_type IN  
  ('in-situ', 'remote'))  
);
```



```
CREATE TABLE quality(  
    quality_id INTEGER NOT NULL,  
    obs_id INTEGER NOT NULL,  
    quality_name VARCHAR (50),  
    quality_type VARCHAR (50),  
    quality_Value REAL,  
    CONSTRAINT quality_pkey PRIMARY KEY (quality_id),  
    CONSTRAINT fk_quality FOREIGN KEY (obs_id) REFERENCES  
    observations(obs_id)  
    );
```

```
CREATE TABLE sensorinfo(  
    sensor_id INTEGER NOT NULL,  
    spatialcoverage VARCHAR (100),  
    temporalcoverage VARCHAR (100),  
    unit VARCHAR (30),  
    valuerange VARCHAR (30),  
    collectionfrequency VARCHAR (50),  
    collectionmethod VARCHAR (50),  
    CONSTRAINT sensorinfo_pkey PRIMARY KEY (sensor_id),  
    CONSTRAINT fk_sensorinfo FOREIGN KEY (sensor_id) REFERENCES  
    sensor(sensor_id)  
    );
```

```
CREATE TABLE observations (  
  obs_id integer NOT NULL,  
  pheno_id integer NOT NULL,  
  covid integer NOT NULL,  
  sensor_id integer NOT NULL,  
  foi_id integer NOT NULL,  
  obsv_type character varying(50),  
  CONSTRAINT obsv_pkey PRIMARY KEY (obs_id),  
  CONSTRAINT fk1_obsv FOREIGN KEY (pheno_id)  
REFERENCES observed_phenomenon(pheno_id),  
  CONSTRAINT fk2_obsv FOREIGN KEY (covid) REFERENCES  
coverage(covid),  
  CONSTRAINT fk3_obsv FOREIGN KEY (sensor_id)  
REFERENCES sensor(sensor_id),  
  CONSTRAINT fk4_obsv FOREIGN KEY (foi_id) REFERENCES  
feature_of_interest(foi_id)  
);
```

```
CREATE TABLE in_situ_sst(  
temp_id INTEGER NOT NULL,  
obs_id INTEGER NOT NULL,  
time_stamp date,  
value VARCHAR (30),  
CONSTRAINT in_situ_sst_pkey PRIMARY KEY (temp_id),  
CONSTRAINT fk_in_situ_sst FOREIGN KEY (obs_id)  
REFERENCES observations(obs_id)  
);  
  
SELECT AddGeometryColumn( 'public', 'in_situ_sst',  
'the_geom', 4326, 'POINT', 2 );
```

```
CREATE TABLE in_situ_lst(  
temp_lst_id INTEGER NOT NULL,  
obs_id INTEGER NOT NULL,  
time_stamp date,  
value VARCHAR (30),  
CONSTRAINT in_situ_lst_pkey PRIMARY KEY  
(temp_lst_id),  
CONSTRAINT fk_in_situ_lst FOREIGN KEY (obs_id)  
REFERENCES observations(obs_id)  
);  
  
SELECT AddGeometryColumn( 'public', 'in_situ_lst',  
'the_geom', 4326, 'POINT', 2 );
```

```

CREATE TABLE in_situ_ret(
  ret_id INTEGER NOT NULL,
  obs_id INTEGER NOT NULL,
  time_stamp date,
  value VARCHAR (30),
  CONSTRAINT in_situ_ret_pkey PRIMARY KEY (ret_id),
  CONSTRAINT fk_in_situ_ret FOREIGN KEY (obs_id)
  REFERENCES observations(obs_id)
);

SELECT AddGeometryColumn( 'public', 'in_situ_ret',
'the_geom', 4326, 'POINT', 2 );

```

Appendix B

Table population SQL statements:

insert into coverage

Values (1, 'in-situ-1st', 'vector', 1, 'A vector coverage of time series land surface temprature observations from in-situ sensor within Colorado State in the USA');

insert into coverage

Values (2, 'remote-1st_day', 'raster', 1, 'A raster coverage of time series land surface temprature observations during the day within Colorado State in the USA');

insert into coverage

Values (3, 'in-situ_ret', 'vector', 2, 'A vector coverage of time series reference evapotranspiration observations from automatic weather station within Nebraska State in the USA');

insert into coverage

Values (4, 'in-situ_sst', 'vector', 3, 'A vector coverage of time series sea surface temperature observations within the Gulf of Mexico sea');

insert into coverage

Values (5, 'remote-lst_night', 'raster', 1, 'A raster coverage of time series land surface temperature observations during the night within Colorado State in the USA');

insert into coverage

Values (6, 'remote-lst_day_utm', 'raster', 1, 'A raster coverage of time series land surface temperature observations during the day within Colorado State in the USA in utm coordinate system');

insert into coverage

Values (7, 'ndvi', 'raster', 4, 'A raster coverage of time series Normalised Difference Vegetation Index NDVI observations within Nebraska State in the USA');

insert into coverage

Values (8, 'SRTM-DTM', 'raster', 5, 'A raster coverage of time series elevation data from SRTM observations within Colorado State in the USA');

Insert into feature_of_interest

Values (1, 1, 'LST', 'land surface temperature from in-situ sensors', 1);

Insert into feature_of_interest

Values (2, 2, 'LST_day', 'land surface temperature during the day from modis satellite', 1);

Insert into feature_of_interest

Values (3, 3, 'RET', 'Reference Evapotranspiration from automatic weather stations', 2);

Insert into feature_of_interest

Values (4, 4, 'SST', 'sea surface temperature from in-situ sensors', 3);

Insert into feature_of_interest

Values (5, 5,'LST_night', 'land surface temperature during the night from modis satellite sensor', 1);

Insert into feature_of_interest

Values (6, 6,'LST-day_utm','land surface temperature during the day in utm coordinate system from modis satellite ', 1);

Insert into feature_of_interest

Values (7, 7,'NDVI', 'Normalised Difference Vegetation Index from Modis satellite sensor', 4);

Insert into feature_of_interest

Values (8, 8,'Elevation', 'surface terrian elevation observation from SRTM satellite', 5);

Insert into observed_phenomenon

Values (1,'LST', 'Land Surface Temperature observations', 'Colorado area in the USA');

Insert into observed_phenomenon

Values (2,'RET', 'Reference Evapotranspiration from automatic weather sations', 'Nebraska area in the USA');

Insert into observed_phenomenon

Values (3,'SST', 'Sea Surface Temperature observations', 'Gulf of Mexico sea near the US');

Insert into observed_phenomenon

Values (4,'NDVI', 'Normalised Difference Vegetation Index from Modis satellite sensor', 'Nebraska area in the USA');

Insert into observed_phenomenon

Values (5,'Elevation', 'Surface Terrian Elevation observations from SRTM satellite', 'Colorado area in the USA');

insert into sensorplatform

Values (03060, 'NA', 4, -107.693, 38.544, 'Colorado USA', 'MONTROSE 11 ENE');

insert into sensorplatform

Values (03079, 'NA', 4, -106.171, 38.099, 'Colorado USA', 'SAGUACHE 2 WNW');

insert into sensorplatform

Values (03084, 'NA', 4, -106.144, 37.707, 'Colorado USA', 'CENTER A 4 SSW');

insert into sensorplatform

Values (53005, 'NA', 4, -106.122, 38.811, 'Colorado USA', 'BUENA VISTA 2 SSE');

insert into sensorplatform

Values (4000, 'NA', 1, -96.933, 40.933, 'Nebraska USA', 'Beatrice');

insert into sensorplatform

Values (4001, 'NA', 1, -97.967, 41.15, 'Nebraska USA', 'CentralCity');

insert into sensorplatform

Values (4002, 'NA', 1, -98.133, 40.567, 'Nebraska USA', 'Clay Center');

insert into sensorplatform

Values (4003, ': R332', 12, -88.793, 26.998, 'Gulf of Mexico', 'NOAA Ship Oregon II');

insert into sensorplatform

Values (4004, ': R332', 12, -88.816, 27.0252, 'Gulf of Mexico', 'NOAA Ship Oregon II');

insert into sensorplatform

Values (4005, ': R332', 12, -88.843, 27.047, 'Gulf of Mexico', 'NOAA Ship Oregon II');

insert into sensor

Values (1, 3060, 'in-situ', '03D152');

insert into sensor

Values (2, 3079, 'in-situ', '0AA19E');

insert into sensor

Values (3, 3084, 'in-situ', '0C644E');

insert into sensor

Values (4, 53005, 'in-situ', '1057D6');

insert into sensor

Values (5, 4000, 'in-situ', 'NA');

insert into sensor

Values (6, 4001, 'in-situ', 'NA');

insert into sensor

Values (7, 4002, 'in-situ', 'NA');

insert into sensor

Values (8, 4003, 'in-situ', 'NA');

insert into observations

Values (1, 1, 1, 1, 1, 'in-situ');

insert into observations

Values (2, 1, 1, 2, 1, 'in-situ');

insert into observations

Values (3, 1, 1, 3, 1, 'in-situ');

insert into observations

Values (4, 1, 1, 4, 1, 'in-situ');

insert into observations

Values (5, 1, 1, 1, 1, 'in-situ');

insert into observations

Values (6, 1, 1, 2, 1, 'in-situ');

insert into observations

Values (7, 1, 1, 3, 1, 'in-situ');

insert into observations

Values (8, 1, 1, 4, 1, 'in-situ');

insert into in_situ_lst

Values(1, 1, '2011-08-01',16.1,ST_GeomFromText('POINT(-107.693 38.544)', 4326));

insert into in_situ_lst

Values(2, 2, '2011-08-01',17.5,ST_GeomFromText('POINT(-106.171 38.099)', 4326));

insert into in_situ_lst

Values(3, 3, '2011-08-01',17.7,ST_GeomFromText('POINT(-106.144 37.707)', 4326));

insert into in_situ_lst

Values(4, 4, '2011-08-01',4.5,ST_GeomFromText('POINT(-106.122 38.811)', 4326));

insert into in_situ_lst

Values(5, 5, '2011-08-02',15.5,ST_GeomFromText('POINT(-107.693 38.544)', 4326));

insert into in_situ_lst

Values(6, 6, '2011-08-02',17.4,ST_GeomFromText('POINT(-106.171 38.099)', 4326));

insert into in_situ_lst

Values(7, 7, '2011-08-02',16.6,ST_GeomFromText('POINT(-106.144 37.707)', 4326));

insert into in_situ_lst

```
Values( 8, 8, '2011-08-02',-15.5,ST_GeomFromText('POINT(-106.122 38.811)', 4326));
```

```
insert into in_situ_ret
```

```
Values( 1, 9, '2005-09-09',6.652,ST_GeomFromText('POINT(-96.933 40.30)', 4326));
```

```
insert into in_situ_ret
```

```
Values( 2, 10, '2005-09-10',5.34,ST_GeomFromText('POINT(-97.967 41.15)', 4326));
```

```
insert into in_situ_ret
```

```
Values( 3, 11, '2005-09-11',5.73,ST_GeomFromText('POINT(-98.133 40.567)', 4326));
```

```
insert into in_situ_sst
```

```
Values( 1, 12, '2006-08-01 01:08:07',30.20,ST_GeomFromText('POINT(-88.793 26.998)', 4326));
```

```
insert into in_situ_sst
```

```
Values( 2, 13, '2006-08-01 01:18:07',30.176,ST_GeomFromText('POINT(-88.816 27.0252)',  
4326));
```

```
insert into in_situ_sst
```

```
Values( 3, 14, '2006-08-01 01:28:07',30.148,ST_GeomFromText('POINT(-88.843 27.048)',  
4326));
```

A sample raster table creation and data population using the raster2pgsql.py script.

95



Masters Program in **Geospatial Technologies**

HETEROGENEOUS SENSOR DATABASE FRAMEWORK FOR THE SENSOR
OBSERVATION SERVICE: INTEGRATING REMOTE AND IN-SITU SENSOR
OBSERVATIONS AT THE DATABASE BACKEND

MADUAKO IKECHUKWU